

برمجة الانترنت باستخدام لغة جافا



تأليف

الدكتورة

جنان عبد الوهاب فيضي

الدكتور

صلاح محمد أمين الخياط

مؤسسة الوراق

عمان - الأردن

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ
"وَمَا أُوتِيتُمْ مِنَ الْعِلْمِ إِلَّا قَلِيلًا"
صدق الله العظيم

برمجة الأنترنيت باستخدام لغة جافا

برمجة الأنترنت باستخدام لغة جافا

Internet Programming Using JAVA

BIBLIOTHECA ALEXANDRINA

الدكتورة

تأليف

الدكتور

جنان عبد الوهاب فيضي

Dr. Jinan A. W. FAIDHI
Ph D, MBCS, VMACM
Associate Professor

صباح محمد أمين الخياط

Dr. Sabah M.A. MOHAMAD
Ph D, MBCS, MIEE, VMACM
Associate Professor, Chairman

١٩٩٩

مؤسسة الوراق

عمان - الأردن

رقم الايداع لدى دائرة المكتبة الوطنية

١٩٩٩/٣/٣٠٠

رقم التصنيف	:	٠٠٥,١٣٣
المؤلف ومن هو في حكمه	:	صباح محمد أمين محمد الحياط جنان عبد الوهاب فيضي
عنوان المصنف	:	برمجة الانترنت باستخدام لغة جافا
الموضوع الرئيسي	:	١. المعارف العامة ٢. لغات الحاسوب - جافا
بيانات النشر	:	عمان: مؤسسة الوراق للنشر والتوزيع
تم إعداد بيانات الفهرسة والتصنيف الأولية من قبل دائرة المكتبة الوطنية		

حقوق النشر والتأليف محفوظة للناسر

مؤسسة الوراق للخدمات الحديثة

شارع الجامعة الأردنية-عمارة العساف - مقابل كلية الزراعة -

ص.ب ١٥٢٧ عمان ١١٩٥٣ الأردن

تلفاكس ٥٣٣٧٧٩٨

مقدمة الكتاب

بحمد الله تم إنجاز كتابنا المنهجي الموسوم " برمجة الانترنت باستخدام لغة جافا " ، حيث كان حصيلة خبرة تخصصية وتدرسية بالإضافة الى خبرة بحثية في مواضيع هندسية البرمجيات وتصميم اللغات . واخيراً ولحاجة العديد من الجامعات والمؤسسات الأكاديمية والشركات لكتاب منهجي وتدريري يغطي أحد أهم المسارات التطورية في علم الحاسوب وبالذات فيما يتعلق ببرمجة شبكة الانترنت التي أصبحت أحد المرافق الحساسة في التطور العام لأي مؤسسة او فرد لما تزوده من معرفة وأساليب تخاطب مع بقية المؤسسات والأفراد المنتشرين على عموم الكرة الأرضية فان هذا الكتاب جاء ليلي هذا الطموح وليضع الدعائم الاساسية لبرمجة الانترنت منطلقين من اسعراض مستلزعات برمجة هذه الشبكة الاساسية ابتداءً من اللغة الارشادية (HTML) وكذلك استخدام نصوص لغة جافا (JavaScript) واخيراً تعمق المؤلفين باستعراض كل جوانب اللغة الاساسية في برمجة الانترنت وهي لغة جافا (Java) وبشكل ركز على الأوجه الحديثة التي جاءت بها هذه اللغة من أساليب تخاطب حديثة وأساليب بناء برمجي يعتمد على البرمجة المرئية (Visual Programming) التي تعبر عنها جافا من خلال تصميم الوحدات التفاعلية المعروفة بالابليت (Applets) .

إن لغة جافا أصبحت في العديد من الجامعات الرصينة الغربية هي لغة البرمجة الاولى والرئيسية واعتبرت بنظر الكثير الباحثين والمبرمجين هي اللغة التي سوف تستخدم لبرمجة نظم الالفية الثالثة ، وانطلاقاً من فلسفة جامعة العلوم التطبيقية فان تدريس كل ما هو حديث وتطبيقي هام للصناعة البرمجية لا بد أن يكون ضمن مناهجها ، ولذا جاء هذا الكتاب منسجماً مع هذه الفلسفة ولكي يكون منهجياً لساق البرمجة الموجهة للكيانات (OOP) ملياً لطموحات هذه الثورة البرمجية الحديثة .

أن نسخة جافا المستخدمة في هذا الكتاب هي لغة جافا القياسية (Java 1.1) المستخدمة من قبل كل الشركات المصنعة لترجمات لغة جافا ، ولقد تم اختيار كل البرمجيات المذكورة في هذا الكتاب في مختبرات القسم التدريسية ونشكر العديد من الاخوة الذين قاموا بمساعدتنا في إتمام هذه الاختبارات وبالذات الاخوة : أيمن الشيشاني و عمر شعاده ، وكذلك نشكر مؤسسة الوراق لأخراج هذا الكتاب بالسرعة الممكنة.

وأخيراً فإن المؤلفين يعتبرون هذا الكتاب نواة لتدريس العديد من المساقات المنهجية المعروفة مثل هندسة البرمجيات وتصميم وبناء اللغات والبرمجة الموجهة للكيانات بالاضافة الى كونه مادة للمشاريع البرمجية الحديثة ومنهج مناسب للدورات التدريبية في مجال برمجة الانترنت واستخدام لغة جافا .

ومن الله التوفيق

المؤلفين

محتويات الكتاب

الصفحة

المحتويات

الفصل الاول :- تقنية التعامل مع لغة جافا :

١١	١,١ الأسس التصميمية للغة جافا
١٤	١,٢ معمارية لغة جافا
١٧	١,٣ مكونات برامج جافا
٢٥	١,٤ أساسيات لغة الارشاد المساندة للغة جافا

الفصل الثاني :- ايعازات نصوص جافا المرافقة للغة الارشادية :

٣٣	٢,١ تمهيد
٣٤	٢,٢ أساسيات نصوص جافا

الفصل الثالث :- ثلاثة طرق للتفاعل مع جافا :

٦٣	٣,١ التفاعل عن طريق الادخال والاخراج
٧١	٣,٢ التفاعل عن طريق النوافذ
٧٦	٣,٣ التفاعل بالاعتماد على الابلت

الفصل الرابع :- تركيبة برامجيات جافا :

٨٧	٤,١ مكونات تركيبة جافا الاساسية
٩٣	٤,٢ العبارات الرئيسية للغة جافا
١٠١	٤,٣ تركيبة أدوات التفاعل مع المستفيد
١٠٢	٤,٤ كميات التفاعل
١٠٤	٣,٤,١ نوافذ التفاعل
١٠٥	٣,٤,٢ قوائم الخيارات

١٠٦	٣,٤,٣ حقول الكتابة
١٠٧	٣,٤,٤ مربعات الاختيارات
١٠٨	٣,٤,٥ قوائم الخيارات النازلة
١٠٩	٣,٤,٦ قوائم ذات مولقة مضية
١١٠	٣,٤,٧ إدارة الفعاليات

الفصل الخامس :- إدارة الفعاليات المترامية :

١١٥	٥,١ مفهوم خطوط التعاون المترامية
١٢٠	٥,٢ اجراءات اخرى للتحكم بالخطوط التنفيذية
١٢٦	٥,٣ التفيد المصدر للخطوط التنفيذية

الفصل السادس :- هياكل البيانات باستخدام جافا :

١٣٥	٦,١ اساسيات هياكل البيانات في جافا
١٣٦	٦,٢ هياكل البيانات الخطية
١٤٣	٦,٣ هياكل البيانات اللاخطية

الفصل السابع :- التعامل مع الحركة في صفحات الويب

١٥١	٧-١ التعامل مع الصور المتحركة
١٦٠	٧,١ الحركة باستخدام التقطيع
١٦٢	٧,٢ الحركة باستخدام الخطوط
١٦٣	٧,٣ الحركة باستخدام الحروف
١٦٥	٧,٤ الحركة باستخدام الارقام

١٦٨	المصادر والمراجع
-----	-------	------------------

الفصل الأول

تهيئة التعامل مع لغة جافا

الفصل الأول

تهيئة التعامل مع لغة جافا

1-1 الأسس التصميمية للغة جافا :

لغة جافا صممت لكي تكون نافذة المستقبل لدخول الألفية الثالثة ، حيث استخدمت فيها كل الأدوات البرمجية الملائمة للتطور البرمجي بالإضافة إلى أساسيات وأدوات البرمجة التقليدية. تم تطوير هذه اللغة من قبل فريق برمجي من شركة مايكروسستمز (Sun Micro Systems) بقيادة جيمز كوزلنك (James Gosling) في عام 1991 حيث أسهمت في البداية بلغة أوك (Oak) وبعدها أعيد تسميتها بـ ألفا (Alpha) وبعد إضافة تطويعات أخرى إليها أعيد تسميتها كذلك إلى بيتا (Beta) وأخيرا وبعد إضافة التطويرات الأخرى سميت بجافا (Java) ، ولقد تم تطوير هذه اللغة لتحقيق الأهداف التالية :-

1_ لغة بسيطة (Simple) :

أما لغة بسيطة سهلة الاستخدام ويمكن بناء برامجها بوضوح وبخطوات سهلة ، هو ليس حلم يصعب تحقيقه، فإذا كانت القطع البرمجية المستخدمة واضحة ويمكن تركيبها الواحدة بالأخرى بسهولة بالإضافة أن كل قطعة يمكن أن تراث صفات من القطع التي تنتمي إليها (Inheritance) فإن كل ذلك يمكن أن يصنع اللبنات الأساسية لبرمجة بسيطة واضحة .

2_ البرامج قابلة للتطوير (Extensible) :

من الضروري أن يتم تطوير البرامجيات بسهولة كلما دعت الحاجة لذلك ، وذلك من خلال سهولة فهم القطع البرمجية وسهولة ترابطها وكذلك سهولة تعميمها من خلال البناء على ما تم ميراثه من قطع برمجية سابقة والإضافة إليها.

3_ لغة تناسب التخاطب من خلال شبكات الحاسوب (Networks) :

أن أنساع شبكات الحواسيب في العالم ، أخذ يتطلب وجود لغات خاصة يمكن أن يتخاطب بواسطتها مستخدمي هذه الشبكات بكل بساطة وسهولة، حيث أن طبيعة التخاطب تتطلب بالإضافة إلى تنوع أساليب التخاطب من نصوص وصور وأصوات وما شاكل ذلك من أوساط مختلفة ، إلى وجود أساليب برمجية كفوءة يمكن أن يتم فيها هذا التخاطب بدون تأخير أو عوائق تقطع أساليب الاتصال بين مستخدمي الشبكات ، ولغة جافا جاءت لتلبي هذا الغرض حيث يكون الاتصال في الشبكات سهلاً وكفوءاً .

4_ البرامج يجب أن تكون قابلة للتنفيذ في أي نظام (Multi-Platform) :

ويقصد فيها إمكانية تنفيذ برامجها حافاً في أي نقطة من نقاط شبكة الحواسيب مهما اختلفت المكونات المادية (Hardware) وكذلك نظم التشغيل في نقاط الشبكة ، وحيث أن مترجم لغة جافا ينتج برنامجاً وسطياً كاملاً يطلق عليه (Byte code) يمكن تنفيذه في أي نقطة من نقاط الشبكة مهما كانت معماريتها ونظام تشغيلها.

5_ برامج تستعمل لبناء صفحات تخاطب في شبكات الحاسوب (Web Pages Design) :

ويقصد بأن لغة جافا يمكنها أن تترجم ما يطلب تخاطبه من خلال صفحات الشبكة التي تسمى بصفحات الويب وذلك من خلال استعمال تقنية استدعاءات مقاطع جافا (الابليت Applets) في داخل برامج اللغة الأساسية الأرشادية(HTML) لبناء البنى التحتية لصفحات الويب.

6_ برامجها يجب أن تبنى وفق أسس هندسة البرمجيات (Plug and Program) :

وهذا يعني بأن برامجها لغة جافا يجب أن تدعم أسس البناء البرمجي المتقدم وبالذات ما يتعلق بأسس البرمجة الموجهة للكيانات (Object Oriented Programming) التي تؤكد أن المقاطع البرمجية يجب أن تكون مستقلة ويمكن إعادة استخدام أي قطعة في أي برنامج جديد بالإضافة إلى إمكانية تطوير أي قطعة من خلال أرث صفات قطع برمجية سابقة والإضافة إليها فقط .

7_ برامجها لها القابلية على بناء أطر تتحاور بينية مع المستخدمين (User Interface):

ويقصد به وجود أدوات يمكن استخدامها لبناء أطر التحوار مع المستخدم ومن خلال استخدام أدوات الرسم (Graphics) وحتى اعتماد أدوات المناخات المتعددة من صوت وصورة وأفلام فيديو (Multimedia) وحتى إمكانية بناء برمجيات الحقيقة المرئية (Visual Reality) .

8_ برامجها يجب أن تكون لها قابلية التنفيذ المتوازي (Built-in Concurrency) :

أن أسلوب التخاطب المعتمد على استخدام رسوم وأصوات ونصوص ونوافذ يجعل من الضرورة توفر أسلوب تنفيذ متوازي ، ولغة جافا يمكن أن تؤمن مثل هذه الطريقة في التنفيذ من خلال تقنيات تسمى بالخيوط (Threads) أو بأسلوب برمجي يعرف بأسلوب تنفيذ المهمات البرمجية (Event-Driven Programming or MultiThreading) .

9_ برامجها لها مزايا للتحوار مع البرمجيات المختلفة الأخرى (Port-Oriented) :

وهذا يعني سهولة اتصال برمجيات لغة جافا مع البرمجيات الأخرى وبالذات مع الأدوات البرمجية الأساسية التي تتعامل مع الصوت والصورة وغيرها من الأدوات ، ولغة جافا يمكنها الاتصال بكل هذه البرمجيات الأساسية من خلال اعتمادها سواقات التقنية المعروفة بـ (Active X) بالإضافة إلى إمكانية اتصالها باللغات القياسية مثل لغة سي ولغة CGI ولغة Perl وغيرها .

10_ برامجها يجب أن تكون آمنة في وسط التحوار الشبكي (Secure Programs) :

أن أمنية التحوار البرمجي في وسط شبكي مفتوح للتحوار من قبل ملايين المستخدمين يعتبر مسألة هامة جداً ، ولقد أعطت لغة جافا اهتماماً خاصاً لهذه الصفة من خلال اعتمادها أساليب برمجية للرجوع للمعلومة في الذواكر المتوفرة في الشبكة بطريقة واضحة وورصنة وتطلب هذا التخلص من تقنية

المؤشرات (Pointers) التي كانت اللغات الاخرى تستخدمها والسبي كانت نقطة الضعف الرئيسية في الحصول على المعلومات بطريقة غير مخطلة .
بالاضافة إلى صفات عديدة أساسية أخرى مثل قابلية جمع المسافات المبعثرة في الذاكرة (Garbage Collection) وإمكانية استخدامها في برمجة الاجهزة النقولية (Mobile Code) وغيرها من المميزات الأساسية الخاصة والعامة ، حيث شكلت كل هذه الصفات الحجر الأساس لاعتماد لغة جافا لبرمجة نظم الآلية الثالثة من هذا العصر .

1-2 معمارية لغة جافا :

أن الأهداف التصميمية التي تم ادراجها في الفقرة أعلاه تفصح عن المكونات المعمارية للغة جافا ، حيث يمكن أن نوصف معمارية لغة جافا على أنها معالج لصفحات شبكات الويب . وهذا يتطلب أن تكون للمعالج القدرة على استرجاع صفحات الويب (Retrieval) المكتوبة بلغة (HTML) وكذلك معالجة الصفحات التي تحتوي مدخلات (Form) من خلال نوافذ نصية (Text Fields) أو كبسات (Buttons) أو قوائم مضمينة (Scroll Bars) أو ما شاكل ذلك ، وتختلف لغة جافا عن الطريقة التقليدية لمعالجة صفحات الويب في شبكة الانترنت حيث أن كل الأوامر كانت تعالج بواسطة ايعازات موجودة في المعالج المركزي (Server) وليس من خلال الاعتماد الأكبر على أوامر موجودة في حاسبة المستخدم (Client) والتي هي أصلاً مكتوبة بلغات مختلفة مثل (PERL) و (TCL) وأن علم الاعتماد على أوامر موجودة في المعالج المركزي يعني تقليل الاتصالات بين المستخدم والمعالج المركزي للشبكة ولذلك نتوقع مع استخدام لغة جافا تحسن كبير في أسلوب التخاطب في شبكات الويب .

بالإضافة إلى كونها معالج لصفحات الويب فإن معمارية جافا يمكن تصورها كذلك على أنها مترجم للغة برمجية من النوع الموجه للكائنات (Object-Oriented) مضافاً إليها صفات مثل التنفيذ المتعدد للواجبات (Multithreading) وبناء واجهات متقدمة للتفاعل مع المستخدمين (User Interfaces) وجمع المسافات المبعثرة في الذواكر (Garbage Collection) ومعالجة الحالات الخاصة (Exception Handling) وبخاصة في المناخات البرمجية التوزيعية أو الشبكية ، ومن جهة أخرى يمكن اعتبار لغة جافا على أنها اطرز برمجي

(Framework) يمكنه التفاعل مع مختلف البرامجيات والتطبيقات البرمجية من خلال وجود مرافق متعددة للتفاعل مع مختلف البرامجيات والتطبيقات البرمجية من خلال وجود مرافق متعددة للتفاعل مما يجعل برامجيات لغة جافا قابلة للتطوير. وفي الواقع أن معمارية لغة جافا تحمل كل هذه الصور التي نحدثنا عنها ، والحجر الاساس في معمارية لغة جافا وجود محور أساسي يسمى بقلب جافا (Java Development Kit) التي تكون اللبنة الأساسية من الابعازات، ومحاطة بمرافق (Ports) يمكنها التخاطب من خلالها مع بقية القطع البرمجية المختلفة التي تسمى (API) وتحتوي معمارية لغة جافا وحدات تمكنها التماور مع بقية التطبيقات البرمجية المختلفة ، وبالذات فإنها تحتوي على وحدة (Active X) التي تمكن جافا التفاعل مع العديد من البرامجيات واللغات المختلفة بطريقة تشبه تقنية (OLE) المعروفة في وصل البرامجيات وكذلك وحدة (Open Doc) التي تساعد في استرجاع وعرض صفحات الويب وأخيرا وحدة (Live Connect) التي تساعد في عمليات التفاعل في الوسط الشبكي ، وفي الواقع أن هناك قطع برمجية عديدة أصبحت قياسية وتستخدم على الدوام مع المحور الاساسي للغة جافا مثل (JDBC API) والتي تساعد في التماور مع قواعد البيانات الموزعة في الشبكة و(IDL API) التي تساعد في التماور مع التطبيقات القياسية الاخرى المعروفة بـ (CORBA) و(RMI API) التي تساعد في كتابة أي برنامج يتفاعل بطريقة موزعة و (JTAPI) التي تساعد في اعتماد التفاعل التلفوني من خلال لغة جافا و (Speech API) التي تساعد في تزويد لاجراءات للتعامل مع التطبيقات الصوتية وغيرها من القطع البرمجية المختلفة التي لا يسعنا هنا ذكرها بالتفصيل . ويتكون قلب لغة جافا من المكونات التالية :

أ - مترجم لغة جافا (javac)

ب - مفسر لغة جافا (java)

ج - محول برامجيات جافا إلى ملفات (HTML) (javadoc)

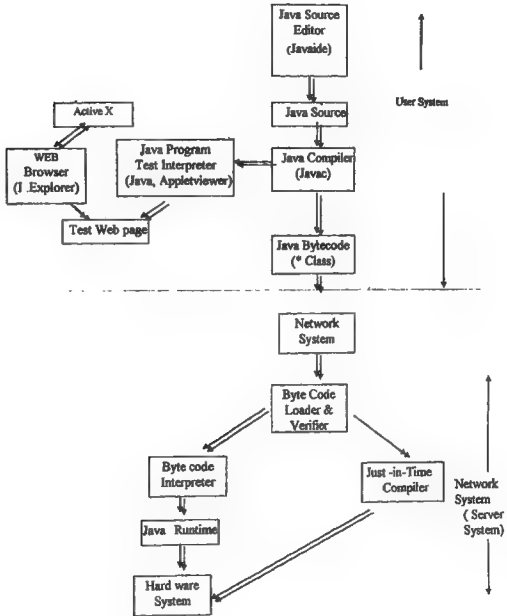
د - محول برامجيات جافا المترجمة (Byte code) إلى صيغة مقروءة (javap)

هـ - مدقق برامجيات لغة جافا (jdp)

و - فاحص الابليت (applet viewer)

٣- تحول برامجيات جافا إلى لغة C (javah)

والشكل رقم (1) يوضح مراحل تنفيذ برامجيات لغة جافا والتي تمر ببعض مستويات المعمارية التي تحدثنا عنها سابقاً



شكل (1) مراحل تنفيذ برامجيات لغة جافا

وبالإضافة الى المكونات البرمجية الموجودة في قلب لغة جافا فأن المبرمج يحتاج الى وحدات تكميلية مثل:

1. مجموعة من الصور توضع في موقع خاص بها (Directory) باسم مثل (C:\images) ويجب أن تكون الصور من الانواع (gif) او (gpj) ، ويفضل تصنيف الصور الى صور نافذة لكي تكون خلفية (Background) او صور للاستخدام المباشر (foreground) .
2. مجموعة من ملفات الاصوات توضع في موقع خاص بها باسم مثلا (C:\audio) ويجب أن تكون من الانواع (au) او (wav) .
3. تحميل معالج صفحات الويب (WEB Browser) الذي يجب أن يكون له القابلية على معالجة نصوص لغة جافا (JavaScript) مثل (Microsoft Explorer Version4) .
4. تحميل معالج لكتابة نصوص لغة (HTML) مثل : AceExpert او ما شاكل ذلك .
5. تحميل برامجيات معالج الاصوات من نوع (Waveedit) او اي برامجيات مكافئة.
6. تحميل برامجيات معالجة مسالدة المعروفة بـ (Active x) .
7. تحميل برامجيات لتحويل صيغ الصور مثل (Graphics Work Shop) .
8. تحميل معالج لكتابة نصوص لغة جافا مثل (Javaide) او اي معالج نصوص آخر .

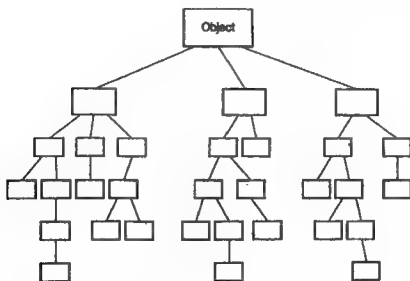
1-3 مكونات برامج جافا :-

لغة جافا لغة أساس بنائها عبارة عن وحدات برمجية مستقلة تسمى بالقطعة المصنفة (Class) ، وهذه الوحدات البرمجية تولد عناصر تنفيذية عند تفعيلها بالذاكرة تسمى بالوحدات التنفيذية (Object) ، والبرنامج الذي تقوم بكتابته يمكن أن يولد مجموعة

من الوحدات التنفيذية التي عند تفاعلها وتخطبها مع بعض تولد النتائج المطلوبة ، أن القطع المصنفة يمكن أن ترث مجموعة صفات من قطع مصنفة أعلى منها مستوى وفي هذه الحالة يطلق على القطعة المصنفة بأنها قطعة مصنفة جزئية (Sub Class) أو يمكن أن تولد من قطعة مصنفة مجموعات جزئية مصنفة ، وفي هذه الحالة تعتبر القطعة المصنفة الاصل بأنها قطعة مصنفة فوقية (Super Class) .

وترث القطعة المصنفة الجزئية نوعين من الصفات :-

- 1_ ترث متغيرات القطعة المصنفة الفوقية (Inherits Variables) .
 - 2_ ترث الاجراءات المصروفة في القطعة المصنفة الفوقية (Inherits Methods) .
- ويمكن تصور علاقات التوارث بين القطع المصنفة المختلفة على أنها شكل شجري يمكن تتبعه في نقل الصفات الموروثة (شكل 2) .



شكل (2) :- الشكل الشجري لعلاقة توارث صفات القطع البرمجية المصنفة .

وتتكون القطعة المصنفة من جزئين رئيسيين :-

1_ جزء التصريحات Declaration Part

2_ جزء الوصف البرمجي Class Body Part

ويتكون جزء التصريحات من المكونات المتسلسلة التالية :-

**[modifiers] class <class name> [extends <superClass>
[implements interface]**

أن التصريحات التي تم وضعها في أقواس مربعة ([]) هي اختيارية ، ولأجل توضيح الدلالة لأجزاء التصريحات نذكر وصفا سريعا عنها وسوف نتاول بالتفصيل من خلال الامثلة استخدامها الدقيقة :-

(1) نوع القطعة المستخدمة (Modifiers):-

وهذه العلاقة التصريحية تحدد نوع القطعة المصنفة ويمكن استخدام للثلاثة علامات تصريحية هي نوع العلاقة العامة الاستخدام (public) أو القطعة التي لا تحتوي على أي قطع مصنفة جزئية (final) ، وإذا لم يذكر أي من النوعين السابقين فإن القطعة المصنفة تعتبر من النوع التعريفي (abstract) .

(2) اجراءات التخاطب مع القطعة المصنفة (Interfaces) :-

ويتم فيها تحديد المتغيرات والاجراءات التي تستخدمها القطعة المصنفة للتخاطب مع القطع الاخرى .

أما الجزء الثاني للقطعة المصنفة ما عدا جزء التصريحات فيعرف بالجزء الوصفي البرمجي للقطعة المصنفة (Class Body) حيث تصرح في داخله المتغيرات المستخدمة (Variables) بالاضافة إلى الاجراءات المستخدمة (Methods) داخل المصنفة ، والمتغيرات التي يتم تصريحها يمكن تقسيمها بشكل عام إلى قسمين رئيسيين :-

أ - متغيرات لا تنتمي للحالة العامة للمصنفة (Non-Member Variables)
مثل المتغيرات الداخلية (local) والمتغيرات المستخدمة في تبادل البيانات (parameters) .

ب - متغيرات تنتمي إلى الحالة العامة للمصنفة (Member Variable)
وهي المتغيرات التي تدخل مباشرة في وصف الحالة العامة للمصنفة البرمجية وهي تصرح داخل الجزء البرمجي للمصنفة وليس داخل اجراءاتها ، ولها الشكل العام التالي :-

[access Specifier] [static] [final] <type> <variable-name>

وهناك محددات الوصول للمتغير ذات أنواع مختلفة (access specifier) وهي :

(1) (private) وهنا المتغيرات لا تستخدم من قبل المصفقات الجزئية التابعة

للمصفقة المصرح فيها المتغير .

(2) (private protected) وهنا يسمح فقط باستخدام المتغير من قبل المصفقة

والمصفقات الجزئية التابعة لها .

(3) (protected) وهنا يسمح باستخدام المتغير داخل المصفقة وأجزائها وكذلك

الحزم التابعة لها (packages)

(4) (public) وهذا يعني استخدام المتغير في كل مكان .

(5) () وإذا لم يذكر أي من العلامات أعلاه فإن ذلك يعني استخدام المتغير

فقط داخل المصفقة والحزم البرمجية التابعة لها .

أما كون المتغير هو من نوع static فإن ذلك سوف يحدد هل أن المتغير هو متغير

للمصفقة (class variable) أم أنه نسخة من متغير المصفقة (instance variable)

وعملية التمييز بين الاثنين تتم إذا وضعت كلمة (static) قبل النوع (type) فهذا يعني أن

المتغير من نوع متغير المصفقة والذي يصرح لمدة في حياة المصفقة ولكن إذا لم تكن كلمة static

قبل النوع فهو يمثل نسخة من متغير المصفقة الذي يصرح وتحجز له مكان في الذاكرة كلما

استخدم . أما إذا جاءت عبارة final قبل كلمة النوع (type) فإن هذا يعني أن المتغير

سيأخذ قيمة ولكن تغير بعد ذلك (constant variable) فعلى سبيل المثال عند ذكرنا :-

```
final static double pi = 3.14159265359 ;
```

```
final static string hello = "Hello World" ;
```

فإن المتغير pi و hello سيأخذان قيمة أولية ولا يمكن تغييرهما بعد ذلك .

وهناك متغيرات يطلق عليها بظلال المتغيرات (shadowed variables) وهذه المتغيرات تصرح في داخل المصنفة الجزئية (subclass) ويوجد في المصنفة الرئيسية متغيرات تحمل نفس الاسم وقد تكون من نوع مختلف وهنا تستخدم ظلال المتغيرات متى ما تم الرجوع إلى المصنفة الجزئية . وأخيرا هناك طريقتين للرجوع إلى المتغيرات نسبة إلى المصنفة المصرح بها ، فإذا استخدمنا < variable name > . this فإن كلمة this تقوم بالإشارة إلى المصنفة التي يعود إليها هذا المتغير وبخاصة عند الإشارة إلى متغير موجود في (subclass) أي مصنفة جزئية ، أما إذا كنا نشير إلى متغير يعود إلى مصنفة فوقية (Super)

(Class) فنضع < Variable name > . super .

أما الجزء الثاني من الجزء البرمجي للمصنفة فهو الاجراءات (Methods) وهذه الاجراءات عبارة عن دوال (functions) وتتكون من ثلاثة أجزاء :-

1_ نوع القيمة العائدة من الدالة (Return Type) .

2_ أسم الاجراء (Method Name) .

3_ الجزء الوصفي البرمجي للإجراء (Method Body) .

وبشكل عام يمكن التعرف على الادوات المستخدمة في تصريح نوع الاجراء كالتالي:-

```
[ access Specifier ][static ] [abstract ] [final ] [ native ]
[synchronized]
return Type < method-name > ( [parameter - list ] )
[throws exceptionlist ] {
...../* method body */
}
```

وإذا كان الاجراء لا يعيد أي قيمة فإنه يجب ذكر عبارة (void) في مكان نوع القيمة المعقدة (return Type) أما إذا كان الإجراء يرجع قيمة فيجب وضع نوع القيمة قبل أسم الإجراء
فمثلا :

```
int sizeup (...){
.....
return ( anIntegerVariable)}
```

ويجب استخدام عبارة الإعادة (Return) داخل الاجراء لإعادة القيمة بواسطة متغير من نفس النوع المصرح .

والإجراءات تقسم إلى نوعين رئيسيين :-

1_ إجراءات تابعة للمصنفة (Class Method)

2_ إجراءات تابعة لنسخ المصنفة (Instant Method)

وفي الواقع أن أكثر الإجراءات هي إجراءات تابعة لنسخ المصنفة إلا إذا تم ذكر عبارة (static) (في بداية تصريح الإجراء مثل :-

```
static void < methodName > ( .. ) {
```

```
...
}
```

وإذا تم استخدام عبارة (final) في تصريح الإجراء فإن ذلك يعني بأن هذا الإجراء لا يمكن تجاوزه (override) من قبل إجراءات أخرى تصرح بنفس الاسم مثلاً . وأحياناً يطلق على الإجراء بالإجراء البنائي (Method Constructor) إذا كان اسمه هو نفس اسم المصنفة (class) المصرح داخلها ، ويستخدم الإجراء البنائي لتهيئة (initialize) متغيرات أو منسلاذة بعض الاجراءات الأساسية لتهيئة عمل الإجراء . ويمكن أن تكون بعض الإجراءات تقوم بإنهاء عمل القطعة المصنفة ويمكن تسمية هذا الإجراء بـ (finalize) ، ويقيد مثل هذا الإجراء لإنهاء الكثير من العمليات أو لإزالة بعض المتغيرات أو لعلق بعض الملفات التي تم فتحها واستخدامها داخل المصنفة البرمجية . والإجراء يمكن أن ينفذ حالات خاصة تنتج نتيجة أخطاء في القراءة أو ما شاكل ذلك (Method Exceptions) فمثلاً الإجراء التالي يؤشر حالة الخطأ عندما نريد القسمة على صفر :-

```
public static int mydivide (int x , int y ) throws ArithmeticException
if ( y == 0 )
    throw new ArithmeticException( );
else
    return ( x / y);
}
```


وعند حاجتنا لأكثر من خطأ يمكننا اقتصاصه فنذكر اسماء الأخطاء الواحد ووضع فاصلة بينهما (comma). والمصنفات البرمجية (classes) يمكن رزماها في حزم يطلق عليها (Packages) وحزم المصنفات هي على نوعين ، فهناك حزم مصنفات مبنية عامة تضم كل من :

- 1_ حزمة (applet) لإخراج صفحات على الويب .
- 2_ حزمة (awt) لإخراج الرسومات والنوافذ .
- 3_ حزمة (io) للإدخال والإخراج الاعتيادي .
- 4_ حزمة (lang) حزمة من برمجيات جافا المفيدة .
- 5_ حزمة (net) حزمة من الإجراءات المفيدة للتخاطب الشبكي .
- 6_ حزمة (util) تمثل حزمة من الإجراءات المساعدة .

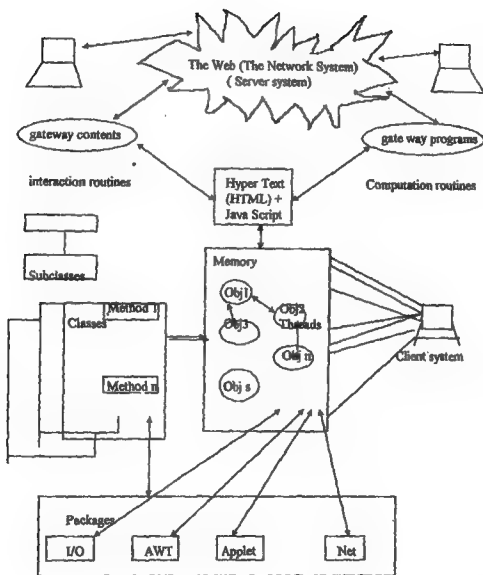
ويمكن استدعاء أي حزمة برمجية من خلال استخدام عبارة (import) فعلى سبيل المثال :-

import java . awt . * ;

ويمكن أن نخلق حزم لمصنفات نقوم ببرمجتها وكل ما نحتاجه إلى ذكر عبارة حزمة (Package) مع اسم الحزمة قبل أي مصنفه نريدها أن توضع في هذه الحزمة البرمجية ، مثلا :

```
package mypackage ;
class my class ( ) {
...}
```

حيث سوف يتم اضافة المصنفه (myclass) اوماتيكيا داخل الحزمة (mypackage) وان مجموع المتغيرات المصرحة داخل المصنفه والاجراءات المستخدمة سوف تعرف التصرف الديناميكي للمصنفه ، وفي هذه الحالة يطلق على التصرف الديناميكي للمصنفه (object) ويمكن آن يتعاون تصرف ديناميكي لمصنفه مع تصرف ديناميكي لمصنفه اخرى ويمكن برمجة هذا التعاون من خلال استخدام ما يعرف بخيوط تعاون التصرف الديناميكي للمصنفات (Threads) ، والشكل رقم (3) يوضح التركيبات الاساسية لمكونات لغة جافا ، ويلاحظ أن لغة جافا يمكن أن تستخدم للتعاور مع المستخدمين الاخرين في شبكة الويب من خلال استخدام البرامج المترجمة (Compiled Classes) في داخل نصوص لغة شبكة الويب الاساسية المعروفة بـ (HTML) التي هي لغة اولية بسيطة تستخدم لاخراج صفحات على الويب (WEB PAGES) .



شکل (3) : مکونات پرامیج لافہ جانا .

1-4 أساسيات لغة الإرشاد المساندة لجافا (HTML) :

هي مجموعة إجراءات غير معتمدة على معيارية معينة (Platform Independent) يمكن استبدالها من خلال وجود علامات معينة (tags) والغاية من هذه الإجراءات هي لإصدار صفحات على شبكة الويب الواسعة تحوي على المكونات مختلفة مثل النصوص والصور والاصوات وما شاكل الى ذلك من انواع المعلومات (Multimedia) ، ومجموعة الإجراءات هذه تشكل لغة للتعريف وثيقة او صفحة على شبكة الويب (HTML) ، وباللغة الإنجليزية يعني ذلك (HyperText Markup Language) ، ويمكن تحرير برمجيات هذه اللغة باستخدام أي محرر ، وهناك محررات خاصة لكتابة برمجيات (HTML) مثل (WYSIWYG) . ولغة (HTML) هي عبارة عن استخدام علامات ترشد البرنامج الذي يقوم بإصدار صفحات الويب (Browser) على أنواع البيانات المستخدمة في نصوص بأنواعها قوائم (lists) ومقاطع نصية (Text Paragraphs) وجداول (Tables) وعناوين (Heads) بالإضافة إلى الصور المستخدمة (images) سواء في متن النصوص أو في الخلفية (Background) مع الأصوات (Voices) أو المؤثرات الصوتية المرافقة للنصوص، وتختلف لغة الإرشاد (HTML) عن لغة جافا بكونها لا تفرق بين الحروف الكبيرة والصغيرة (not case sensitive) ولذلك فإن كتابتنا باستخدام لغة الإرشاد تتكون من ثلاثة مقاطع :-

1_ عنوان الصفحة (Head) .

2_ عنوان النص (Title) .

3_ جسم النص (Body) .

ويمكن توضيح هذا الإجراء بالإطلاع على أبسط برنامج مكتوب بلغة الإرشاد (HTML) ، حيث نعطي عنوان الصفحة (Simple HTML Program) وعنوان النص (HTML)

Example 1) وفي جسم النص نطبع فقرة (Paragraph) نقول أن لغة الإرشاد

بسيطة:-

```
< HTML >
< HEAD >
< TITILE > A Simple HTML Program < / TITILE >
< / HEAD >
< BODY >
    < H1 > HTML Example 1 < / H1 >
    < P > HTML is easy to learn ... OK < / P >
< / BODY >
< / HTML >
```

ويمكن طباعة أي عدد من السطور داخل الفقرة الواحدة وكذلك يمكن عمل الكثير من الإجراءات على النص داخل الفقرة :-

- 1_ جعل أي عدد من الكلمات مائلة (italic) باستخدامها بين `< I > .. < / I >`
- 2_ جعل أي عدد من الكلمات غامقة (Bold) باستخدامها بين `< B > .. < / B >`
- 3_ جعل عدد من الكلمات تحتها عطف (Underline) بوضعها بين `< U > .. < / U >`
- 4_ جعل عدد من الحروف علوية (Superscript) بوضعها `< SUP > .. < / SUP >`
- 5_ جعل عدد من الحروف تحتية (Subscrip) بوضعها `< SUB > .. < / SUB >`
- 6_ جعل عدد من الكلمات عليها علامة عطف (Strike) بوضعها بين `< strike > .. < / strike >`

7_ يمكن كتابة أي نص بحجم كبير أو صغير (Big or Small) من خلال استخدام عبارة `< blockquote >` فعلى سبيل المثال :-

```
< blockquote > < big > This is a big text < / big >
< p >
    < small > This is a small text < / small >
< / p >
< / blockquote >
```

8_ ويمكن التحكم بأي حجم للنص باستخدام زيادات بأي رقم نشاء على الحجم الموجود من خلال `< FONT SIZE = + 2 > ... < / FONT >`

أما فيما يخص القوائم فتتوفر منها أنواع هي التالية :-

أ_ قوائم ذات علامات (Unordered list) : وصيغتها العامة هي :

```
< UL type = square >
```

```
< li > .....
```

```
< li > .....
```

```
:
```

```
< li > .....
```

```
< / UL >
```

وتتكون القوائم إما بعلامة مربع (Square) أو قرص (disc) أو دائرة (circle)

ب_ قوائم مرقمة (Ordered list) : وصيغتها العامة هي :

```
< OL type = a >
```

```
< li > ...
```

```
< li > ...
```

```
< li > ...
```

```
< / OL >
```

وقد نستخدم بدل (a) أرقام مثل (1) أو حروف مثل (i) .

ج_ قوائم غير مرقمة (List without Mark) : وصيغتها العامة هي :

```
< DL >
```

```
< DT > HEAD 1 < / DT >
```

```
< DD > This is a definition of HEAD 1 < / DD >
```

```
< DT > HEAD 2 < / DT >
```

```
< DD > This is a definition of
```

```
HEAD 2 < / DD >
```

```
< / DL >
```

وهنا نوضح أي عناوين أو كلمات كمراس للقائمة .

د- قوائم كبسات (Buttons Menu) :

والكبسات يمكن ان نعتبرها صور لأستخدامها للانتقال إلى معلومات من مكان إلى آخر والمثلل التالي يوضح قائمة فيها عدد اثنين من الكبسات يتم بواسطة اختيار أي منها الانتقال إلى برنامج آخر نوع HTML :

```
<p>
< a href = "prog1.html" < img src = "butt1.gif"
      alt = "first Push" border = 0>
    < / a >
    < a href = "prog2.html" < img src = "butt2.gif"
      alt = "second push" border = 0>
    < /a>
< / p >
```

9- ويمكن التعامل مع الصور (image) داخل برامج لغة الارشاد كالتالي :-

أ- استدعاء الصورة لكي تكون خلفية للنص (Background)

ويتم استدعاء الصورة في داخل تصريح body كالتالي :

```
< body background = "imageName . gif" >
ب- استدعاء الصورة في حيز مستقل داخل صفحة الويب ( Alone )
```

```
< p > < img src = "imageName . gif" > < / p >
ج- استدعاء الصورة إلى يمين أو يسار أو وسط نص ( In text )
```

والمثال التالي يوضح استدعاء صورتين ووسطهما نص :

```
< p align = center >
  < img src = "first . gif" align = right hspace = 4 >
  < img src = "second . gif" align = left hspace = 4 >
.... Any text here ....
< / p >
```

10_ التعامل مع الاصوات (voices) :

ويتم بطريقة مشابهة للتعامل مع الصور والفارق سيكون بنوع الملف المستدعى

أ_ استدعاء الصوت كخلفية (Background)

```
< bgsound src = "< voice file >" loop = "1" >
```

ب_ استدعاء الصوت في أي موقع (Calling a Voice)

```
< a href = "avoice . au" >  
< / a >
```

11_ الانتقال بين مقاطع مختلفة للصفحة الواحدة (Same Page Transferee) :

يمكن الانتقال بين مقاطع في نفس الصفحة (web page) باستخدام ما يسمى

المرسى (anchor) والبرنامج التالي يوضح مثالا لانتقال من كل سؤال إلى جوابه والعودة إلى

قائمة الاسئلة في نفس الصفحة :-

```
<HTML>  
<HEAD>  
<TITLE>HTML WEB PAGES SCROLLING </TITLE>  
</HEAD>  
<BODY BGCOLOR=#FFFFFF VLINK=#000080 TOPMARGIN=0  
LEFTMARGIN=0>  
<CENTER><IMG SRC=" BORDER=0 ALT="ScrollingExample">  
<p><FONT FACE=ARIAL SIZE=3><B> WEB Page </B></FONT><BR>  
<P><FONT FACE=ARIAL SIZE=1> Forward Backward </CENTER>  
<FONT> FACE=ARIAL SIZE=2>  
<P><A NAME="Top">  
<UL>  
  <LI><A HREF="# 1">First Question?</A>  
  <LI><A HREF="# 2">Second Question?</A>  
  <LI><A HREF="# 3">Third Question?</A>  
</UL>  
<OL>  
<br>  
<A NAME=" 1"></A>  
<P><LI><B><I> ANSWER FOR FIRST QUESTION ? </i></b><BR>  
Here is my answer for the first question .....<br>  
<A HREF="#Top"><IMG SRC="test.gif">Back to the top.</A>  
<A NAME=" 2"></A>
```

```

<P><LI><B><I> ANSWER FOR 2ndQUESTION? </i></b><BR>
Here is the answer for the second question ... <BR>
<A HREF="#Top"><IMG SRC="test.gif">Back to the top.</A>
<A NAME="3"></A>
<P><LI><B><I> ANSWER FOR 3rdQUESTION? </I></B><BR>
Here is my answer for the third question ....<BR>
<A HREF="#Top"><IMG SRC="test">Back to the top.</A>
</BODY>
</HTML>

```

12_ الانتقال بين صفحات مختلفة (Different Pages Transferee) :

يمكن الانتقال من أي صفحة إلى صفحة أخرى باستخدام نفس أنواع المراسي المستخدمة في الفقرة السابقة ، والمثال التالي يوضح انتقالات من صفحة رئيسية إلى قائمة خيارات لصفحتين من النوع النصي (Textual Pages)

```

<HTML>
<HEAD>
<TITLE>ssheet - demo</TITLE>
</HEAD>
<BODY BGCOLOR="white">
  <H 1>TEXT MENUE HTML PROGRAM</H 1>
  <A HREF="ssht 1.txt">FIRST TEXT</A>
  <P>
    <A HREF="ssht 2.txt">SECOND TEXT</A>
</body>
</HTML>

```

ويمكن اضافة أي إجراء غير قياسي للغة (HTML) باستخدام دوال من لغة تسمى بنصوص جافا (JavaScript)

والفصل الثاني يوضح كيفية استخدام دوال من نصوص لغة جافا لإضافة فعاليات وإجراءات غير قياسية داخل البرنامج الإرشادي .

الفصل الثاني

أيعازات نصوص جافا

المرافقة للغة الأرشادية

الفصل الثاني

أيعازات نصوص جافا المرافقة اللغة الارشادية

JavaScript

1-2- تمهيد :-

أن اللغة المختصرة التي يمكن بواسطتها اضافة قابليات غير مبنية في اللغة الارشادية (html) يطلق عليها بنصوص جافا المرافقة للغة الارشادية (Java Script) او احيانا بلغة الموحا (Mocha) وذلك لكون هذه اللغة قريبة من لغة جافا وتعتبر مجموعة جزئية منها . ويمكن ترجمة نصوص هذه اللغة من قبل برامجيات استعراض صفحات الويب المعروفة بـ (Browsers) مثل (Netscape& Explorer) ولقد وجد تماما ان تعلم ايعازات نصوص جافا يهيئ المبرمج لكي يأخذ خبرة مبكرة لتعلم لغة جافا الاصلية ، واهم فائدة لاياعازات نصوص جافا أنها لغة مبسطة تعمل من خلال لغة الارشاد (HTML) مباشرة ، ولذلك فإن تطوير أي اجراء لا يحتاج منا الخروج لمناخ برمجي آخر ومن ثم العودة ، وكل ما تحتاجه هو وضع مؤشر (Tag) يوضح للغة الارشاد بأن هناك ايعازات من نصوص جافا سوف يتم ادخالها وهذا المؤشر هو <script> ، وعندما تنتهي الايعازات يتم وضع مؤشر النهاية </script> . ويفضل وضع نصوص جافا في بداية برامج لغة الارشاد في داخل <head> ... </head> . زمن المفيد الذكر بأن ايعازات نصوص جافا هي حماسة لنوع الحرف سواء كان كبيرا او صغيرا ، فعلى سبيل المثال البرنامج التالي يستخدم دالة فقط لطبع تحذير معين على صفحة ويب فيها نص بسيط :-

```
<html>
<head>
<title> My first Java Script Program </title>
  <script language = "Java Script">
    document . alert ("Welcome to Java Script");
  </script>
```

```

</head>
<body>
<p> This is my first Java Script Program </p>
</body>
</html>

```

2-2- أساسيات نصوص جافا :-

سوف نستعرض في هذه الفقرة بعض الاساسيات التي تعتمد عليها نصوص جافا في البرمجة :-

أ المتغيرات وانواعها :- تدعم نصوص جافا بشكل رئيسي ثلاثة انواع من المتغيرات وهي :

- 1- الاعداد (Integers & Reals) .
- 2- الخيوط الرمزية (string) .
- 3- البوليانية (Booleans) .

ويمكن تصريح أي متغير في اي نوع من الانواع اعلاه بطريقتين ، الاولى تكون باستخدام كلمة (var) مثل :

```
var daysinyear=365 ;
```

او بدون استخدام كلمة (var) مثل :

```
Age = 28 ;
```

اما اذا تم تصريح متغير بدون وضع قيمة اولية له مثل :

```
var TempVar ;
```

فان نوعه سوف يكون مجهولا لحين استخدامه فمثلا اذا استخدم المتغير في موقع عدد حقيقي فانه سيكون (Real) وهكذا ويمكن خلق متغيرات مركبة مثل المصفوفات (Arrays) من خلال استخدام عداد في داخل قوس مربع بالقرب من اسم المتغير ، فمثلا عندما نصرح

```
arrayname [ index ] ;
```

فان ذلك يعني بأن المصفوفة arrayname هي ذات بعد واحد وعدادها هو في المتغير index ، والبرنامج التالي يقوم بتصريح مصفوفة ذات بعد واحد في دالة اسمها Make Array ومن ثم استدعاء هذه الدالة في دالة أخرى اسمها WriteData التي تقوم بخلق مصفوفة ذات بعد واحد وبأربعة خانات نضع فيها أربع خيوط رمزية (strings) ثم نطبع المصفوفة :

```
<html>
<head>
<script language = "Java Script">
    function MakeArray (n) {
        this . length = n ;
        for (var x = 1 ; x <=n ; x ++ ) { this [x] = 0 }
        return . this ;
    }
    function writeData ( ) {
        var counter ;
        var onearray = new MakeArray (4) ;
        onearray [1] = "Java" ;
        onearray [2] = "and"
        onearray [3] = "HTML" ;
        onearray [4] = "programming" ;
        for (counter = 1 ; counter <=4 ; counter ++ ) {
            document . write (onearray [counter] + " " ) ;}
        }
    writeData ( ) ;
</script>
</head>
<body>
</body>
</HTML>
```

ويمكن تحويل نوع متغير إلى آخر حسب القواعد التالية :

قاعدة (1) :-

string = string + integer

فقطى سبيل المثال :

var Days = "334" ;

var DaysDec = 31 ;

Daysnow = Days + DaysDec ;

فان قيمة DaysNow ستكون " 33431 " ، لأن القيمة الاولى كانت خطيا رمزيا.

قاعدة (2) :-

integer = integer + string

فقطى سبيل المثال :

var DaysDec = 31 ;

var Days = "334" ;

DaysNowI = DaysDec + Days ;

فان قيمة DaysNowI ستكون 365 لأن القيمة الاولى كانت عددا صحيحا .

ب المتراجحات (Expression) :

المتراجحات هي عبارة عن أشباه معادلات تستخدم المتغيرات وبعض الاجراءات (operators & methods) للحصول على قيمة معينة ، والمتراجحات على ثلاثة أنواع :-

- 1- المتراجحات العددية (Arithmetic Expression) .
والتي يكون ناتجها النهائي عددا معينا .
- 2- متراجحات الخيوط الرمزية (String Expression) .
والتي يكون ناتجها النهائي خطيا رمزيا .
- 3- المتراجحات البوليائية (Boolean Expression) .
والتي يكون ناتجها قيمة منطقية (True or False) .

ويمكن تقسيمها كذلك إلى الأنواع التالية حسب طريقة استخدامها :

1- المتراجحات الشرطية :- وهي تلك المتراجحات التي تعمل بطريقة مشابهة للعبارة الشرطية if - then - else وتستخدم الرمز (?) للدلالة على الشرط فمثلا المتراجحة الشرطية التالية :-

timetype = (hour > 12) ? "PM" : "AM" ;

وهنا يتم اختبار الشرط حسب قيمة ما موجود في المتغير hour فإذا كانت القيمة أكبر من 12 فيتم وضع الخيط الرمزي "PM" في المتغير timetype وإذا لم يتحقق ذلك الشرط فيتم وضع قيمة "AM" في المتغير timetype .

2- متراجحات الإزاحة :-

وهي تلك المتراجحات التي تعمل بنفس طريقة عبارات الإزاحة (assignment) ويستخدم لهذا الغرض العمليات (= / أو * أو = - أو + =) فعلى سبيل المثال :-

X = 4 ;	{ وتعني أن قيمة X ستكون 45 }
X * = 10 ;	{ وتعني ضرب X في 10 ثم إعادته إلى X }
X += 10 ;	{ وتعني جمع X مع 10 ثم إعادته إلى X }
X - = 10 ;	{ وتعني طرح X من 10 ثم إعادته إلى X }
X /= 10 ;	{ وتعني قسمة X على 10 ثم إعادته إلى X }

ويمكن أن نجري بعض العمليات بدون استخدام رموز الإزاحة مثل :-

X ++ ;	{ ويعني جمع قيمة 1 من X وإعادته إلى X }
X - - ;	{ ويعني طرح قيمة 1 من X وإعادته إلى X }

3- متراجحات المقارنة :-

وهي تلك المتراجحات التي تستخدم عمليات ورموز خاصة للمقارنة مثل (== أو != أو > أو = < أو < أو >) والتي يكون ناتجها قيمة الصديق أو الكذب مثل :-

stoploop = (counter > 12) ? true : false .

وهناك عمليات منطقية تعمل على القيم البوليانية مثل عملية أو (or) او عملية مع (and) والتي نرسم لهم (//) و (&&) فمثلا عند أخذ :

```
age 1 = true ;
```

```
age 2 = false ;
```

فإن قيمة المتراجحة age1 && age2 ستكون كاذبة .

٤- متراجحات الخيوط الرمزية :-

وهي تلك المتراجحات التي تعمل على الخيوط الرمزية (strings) وبالأذات نستخدم العمليات (= و +=) لدمج الخيوط الرمزية ، فعلى سبيل المثال:

```
part1 = "Java" ;
```

```
part2 = "Script" ;
```

```
part 3 = part1 + part 2 ;
```

فإن قيمة الجزء (part3) سيكون هو " Java Script " .

ج قیود المتغيرات (Object Properties) :

يمكن ربط صفات كل متغير بطريقة تشبه وضع حقول (fields) داخل القيود (Records) ، ويتم ربط الصفات مثل :-

object Name . property ;

ويتم خلق صفات المتغيرات من خلال تعريف دالة خاصة لذلك ، فمثلا لو أردنا خلق متغير اسمه (browser) وفيه صفتين الأولى الاسم (name) والثانية الموقع (platform) ، ويتم بعد خلق الدالة استخدام هذا المتغير ووضع قيمة لصفاته كالتالي :-

```
browser . name = "Netscape 2.0" ;
```

```
browser . platform = "Windows 95 " ;
```

والدالة يمكن خلقها لتعريف المتغير وصفاته كالتالي :-

```
function browser (name , platform) {  
    this . name = name ;  
    this . platform = platform ;  
}
```


د- العبارات البرمجية لنصوص جافا (Statements) :-
 تمتلك لغة نصوص جافا العديد من العبارات البرمجية التي تشابه العبارات المستخدمة في لغة (C)، ولذلك فسنقوم باختصار شرح انواع هذه العبارات ،
 وحيث أن العبارات تقسم إلى عبارات تكرارية (iterative) ، وشرطية (conditional) وعبارات تسلسلية (sequential) .
 1- العبارات التكرارية : وهي على نوعين رئيسين هما (for و while)

والنوع الأول (for) له الشكل القواعدي التالي :-
 for ([initial-expression;] [condition;] [update-expression]) {
 statements ...
 }

ولنأخذ مثالا الدالة Test التي تحتوي على عبارة for :-
 < script language = " Java Script " >
 function test() {
 var string1 = ' < hr align = "center " width = ' ;
 for (var size = 5 ; size <= 100 ; size += 5)
 document . writeln (string 1 + size + ' % " > ') ;
 }
 < / script >

والمثال أعلاه يقوم بطبع أرقام نسب 5% ويزداد تكراريا كل مرة بزيادة 5% أخرى وصولاً إلى 100% . وهناك نوع آخر من العبارة التكرارية (for) التي لها الصيغة القواعدية التالية :-

for (indexvar in objectname) {
 statement ...
 }

وأما العبارة التكرارية من نوع (while) فلها الصيغة القواعدية التالية:-

while (condition) { statement }

وكمثال على هذه العبارة هو المثال التالي :-

```
<html>
< head >
< script language =" JavaScript " >
    var stoploop =false ;
    var counter = 1 ;
while ( ! stoploop ) {
    document.write(counter, " *10=" counter*10,"<br>" );
    counter ++ ;
    stoploop = ( counter > 12 ) ? true : false ;
}
< / script >
< / head >
< body >
< / body >
< / html >
```

ويقوم البرنامج أعلاه بضرب عداد في العدد (10) ولغاية أن نصل بقيمة العداد إلى (12) .

2- العبارات الشرطية : والعبارات الشرطية ذات نوع واحد هي (if) والتي لها الصيغة القواعدية التالية :

if (condition) { statement } [else { statement }]

والمثال التالي يقوم بإصدار صفحة ويب (Web Page) حيث يقرأ اسم الشخص (name) وعمره (age) وإذا كان عمر الشخص أكبر ويساوي (50) سنة فالتنا نعرض المستخدم بأنه لا يصلح وإذا كان أصغر من ذلك فإنه يصلح :-

```

<HTML>
<HEAD>
<TITLE> First Example </TITLE>
<script language = " JavaScript " >
    function Validate ( form ) {
        if ( form.value >= 50 ) {
            alert ( " You do NOT qualify for this policy " ) ;
        }
        else {
            alert ( " You ARE qualified for this policy " ) ;
        }
    }
</script>
</head>
<form>
Name:
    <INPUT TYPE = " text " NAME = " name " >
Age :
    <INPUT TYPE = " text " NAME = " age " >
    <INPUT TYPE = " button " VALUE = " Submit details "
        onClick = "Validate(this.form.age " >
</FORM>
</BODY>
</HTML>

```

3- العبارات التسليمية :-

وهناك عبارات تسليمية كثيرة نذكرها باختصار كالتالي :-

- 1- عبارة الشرح (comment) : وهما على نوعين ، الاولى عبارة شرح على نفس السطر ويستخدم فيها // أو على عدة سطور ويستخدم /* ... */ .

2- عبارة الخروج (break): يمكن باستخدامها للخروج من العبارة التكرارية .

3- عبارة الاستمرار (continue): بها يتم تكرار التنفيذ في العبارات التكرارية .

4- عبارة الاشارة (with) : وهي العبارة التي يمكن الاشارة إلى اسماء المتغيرات وبخاصة المبنية منها ومن ثم يمكن استخدام صفاتها بدون استخدام اسم المتغير الاصيلي وكما في المثال التالي :

```
with ( document ) {  
    writeln ( " This an HTML text " ) ;  
}
```

3-2- المتغيرات القياسية وصفاتها (JavaScript Standard Objects):
أن لغة نصوص جافا تمنح المبرمج بحدود (20) نوع مختلف من المتغيرات القياسية التي لها صفاتها وطرق استخدامها وتنفيذها (properties and methods) ، والجدول رقم (2-1) يوضح المتغيرات القياسية المستخدمة في نصوص جافا ، ويجب التفريق بين المتغير القياسي (Standard Object) وبين الاجراءات القياسية (Standard Method) فمثلاً المتغير القياسي document غير الاجراء القياسي (...) writeln ، علماً بأن الاجراءات القياسية تعمل ليس فقط على المتغيرات القياسية بل على كل انواع المتغيرات فمثلاً :-

```
myString = " JAVASCRIPT ". toLowerCase ( ) ;
```

حيث أن الاجراء القياسي toLowerCase يعمل على تغير الخيط الرمزي ذو الحروف الكبيرة إلى خيط رمزي ذو حروف صغيرة .

المتغير القياسي	استخدامه
1- كيسة Button	< input type = " button " >
2- لفوف checkbox	< input type = " checkbox " >
3- وثائق document	ويستخدم لإعطاء نوع الوثيقة الحالية للمحطة
4- حاويات form	ويستخدم للوصول إلى المتغيرات داخل حاوية قبيبات < form >
5- حسابات math	ويستخدم للتعرف على الثوابت الحسابية وعملياتها
6- مواقع location	ويستخدم على مواقع الإنترنت لمضيفة من خلال نكر موانها (URL)
7- تاريخ مواقع history	ويستخدم للتعرف على هذه المواقع التي تم زيارتها
8- نص Text	ويستخدم للرجوع إلى القصص < input type = " text " >
9- نافذة windows	ويستخدم للرجوع إلى نافذة تم فتحها من قبل Browser
10- كيسة تقديم submit	ويستخدم لمطوية كيسة تقديم < input type = " submit " >

جدول (1-2) بعض المتغيرات القياسية في نصوص جافا .

ولم نذكر بعض المتغيرات القياسية الأخرى لكونها متداخلة مع اللغة الإرشادية (HTML) وأكثرها يعتبر مؤشرات (tags) في داخلها ، والمثال التالي يوضح التداخل والتشابه بـ متغيرات نصوص جافا واللغة الإرشادية (HTML) :

```
< html >
< head >
< title > This a test HTML document < / title >
< / head >
< body bgcolor = " # 0000bb" fgcolor = " # ffff00 " >
< form name = " myform " >
< input type = " text " name = " person " size = 35 >
< / form >
< / body >
< / html >
```

وهنا تضع اللغة الإرشادية بصورة اونوماتيكية للمتغير القياسي document كل من القيم التالية :

```
document . title = " This a test HTML document "
document . fgcolor = "ffff00 "
document . bgcolor = " 0000bb "
document . href = " http : // www.web com/doc.html "
```

ويلاحظ بأن اللغة الارشادية تستخدم نفس المتغيرات القياسية لنصوص جافا وبالعكس ، ويلاحظ أيضاً بأن لكل متغيرات قياسية صفات تأخذ قيم عند تعريف البرنامج ولذلك فمن المفيد دراسة الصفات المرافقة للمتغيرات القياسية .

2-3-1- الصفات المرافقة لمتغير الكبسولة القياسي :-

يمتلك متغير الكبسولة القياسي (button) الصفات المرافقة التالية:

- الاسم (name)

- القيمة (value)

كما يمتلك إجراءات قياسية تعمل عملية مثل :

- () click لتنفيذ عملية الكبس

- () onclick لربط دالة معينة مع الكبسولة فعلى سبيل المثال :

```
< html >
< head >
< title > Button Demo < / title >
< script language = " JavaScript " >
    function ButtonFunction ( ) {
        alert ( " You pressed a button " );
    }
< / script >
< / head >
< form >
    < input name=" MyBotton " type=" button "
        value="press me!"
        onclick = " ButtonFunction ( ) " >
< / form >
< / body >
< / html >
```

ويوضح البرنامج أعلاه كيفية خلق كبسة باسم MyBotton ووضع عنوان لها ! press me وعند كبسها فعلاً تخرج عبارة You pressed a buttonمن خلال تنفيذ دالة في نصوص جافا هي (ButtonFunction).

2-3-2- الصفات المرافقة لمتغير الاختيارات القياسي :

يمتلك متغير الاختيارات القياسي (Checkbox Object) كل من الصفات المرافقة التالية :-

- الاسم (name)

- ثم الاختيار (checked)

- الاختيار البديل (default checked)

وهناك إجراءات مبنية مرافقة لهذا المتغير القياسي هما :

- (click) دالة لتفعيل عملية الاختيار

- (onClick) دالة لربط الاختيار بدالة من نصوص جافا

والمثال التالي يمثل استخدام هذا المتغير القياسي :-

```
< html > < head >
< title > Checkbox Demo < / title >
< script language = " JavaScript " >
    function CBFfunction ( ) {
        alert ( " You checked option 1 " ) }
< / script >
< / head >
< body >
< form >
    < input name = " cb1 " type = " checkbox "
        onClick = " CBFfunction ( ) " > Option 1
    < input name = " cb2 " type = " checkbox " > Option2
< / form >
< / body >
< / html >
```

وهنا نبني قائمة اختيارات من اختياريين هما Option 1 , Option 2
 وعندما نكبس على الاختيار الاول option 1 تظهر لنا العبارة You
 checked option 1 من خلال تنفيذ الدالة في نصوص جافا ()
 .CBFunction

2-3-3- الصفات المرافقة لمتغير الوثائق القياسية :-
 يمتلك متغير الوثائق القياسي (document) كل من الصفات المرافقة
 التالية :-

- الاسم (title)
 - الموقع (location)
 - لون الرابط (a link color)
 - الرابط (anchors)
 - لون الخلفية (bgColor)
 - لون الاملي (fgColor)
 - القوائم (forms)
 - اخر تحديث (lastModified)
 - تاريخ التحميل (loaded Date)
 - الشخص المحول له (referrer)
 - الشخص المساعد (userAgent)
- وهناك العديد من الاجراءات المرافقة لهذا المتغير القياسي :-

- clear () -
- close () -
- open () -
- write () -
- writeln () -
- onLoad () -
- onUnLoad () -

والمثال التالي يوضح استخدام بعض هذه الصفات والاجراءات المرافقة لمتغير الوثائق القياسي :

```
<html>
< head >
< title > Document Demo < / title >
< script language = " JavaScript " >
    < function WelcomeMsg ( ) {
        alert ( " This document requires Netscpe 2.0 " )
    }
< / script >
< / head >
< body bgcolor = " # 0000ff " fgcolor = " # ffff00 "
    onLoad = " WelcomeMsg ( ) " >

< / body >
< / html >
```

2-3-4 - الصفات المرافقة لمتغير القوائم القياسي :-
يمتلك المتغير القياسي للقوائم (form) كل من الصفات التالية :

- الاسم (name) .
- العناصر (elements) .
- الاجراءات (method) .
- الهدف (target) .
- العمل (action) .

وهناك إجراعين مرافقين وهما (submit () , onSubmít ()
والمثال التالي يوضح استخدام هذا المتغير القياسي :-

```

< html >
< head >
< script language = " JavaScript " >
    function formHandler ( ) {
        alert ( " Name : " +
            document . forms[0]. name .value + " \ n " +
            document . forms[0]. email .value );
    }
</script >
</ head >
< body >
< form onSubmit = " formHandler ( ) " >
    Name : < input type = " text " name = " name " size = 35 >
    Email : < input type = " text " name = " email " size = 30 >
    < input type = " submit " value = " Submit Details " >
</ form >
</ body >
</ html >

```

ويوضح هذا البرنامج إنشاء قائمة من حقلين نصيين (two text fields)
 الاول للاسم والثاني للعنوان الالكتروني يقوم عند قراءتهما تخزين قيمتها في
 القائمة بعد الكيس على كيسة submit وتنفيذ الدالة formHandler ()

2-3-5- الصفات المرافقة للمتغير القياسي لكيسة الراديو:-

المتغير القياسي لكيسة الراديو (Radio Button) الصفات التالية :-

- الاسم (name) .
- القيمة (value) .
- الطول (length) .
- الفهرس (index) .
- ثم الاختيار (checked) .
- الاختيار البديل (defaultChecked) .

وهناك بعض الاجراءات القياسية المرافقة ومنها :-

. Click () -

. onClick () -

والمثال التالي يوضح استخدام هذا المتغير القياسي :

```
< html > < head >
< script language = " JavaScript " >
    function FormHandler ( ) {
        if ( document.forms [0].pge [0].checked )
            document.forms [0].result.value = "Poor";
        if ( document.forms [0].pge [1].checked )
            document.forms [0].pge [1].value = "Good";
        if ( document.forms [0].pge [2 ].checked )
            document.forms [0].pge [2].value = " Excellent " ;
    }
</ script >
</ head >
< body >
< form onSubmit = " FormHandler ( ) ">
    < input type = " text " size = 20  name = " result " >
        Please enter a rating:
    < input type = " radio " value = " Poor " name = " pge "
        onClick = " FormHandler ( ) " checked > 1
    < input type = " radio " value = " Good " name = " pge "
        onClick = " formHandler ( ) " > 2
    <input type = "radio" value="Excellent" name="pge"
        onClick = " FormHandler ( ) " > 3
< / form >
< / body >
< / html >
```

2-3-6- الصفات المرافقة لبعض المتغيرات القياسية الأخرى :-

سوف نستعرض بعض المتغيرات القياسية الأخرى وصفاتها المرافقة والتي لم يتم ذكرها في الفقرات السابقة .

أ - متغير التاريخ القياسي (Date) :-

يحتوي متغير التاريخ لاجراءات قياسية فقط وهي :-

getDate () -
getDay () -
getHours () -
getMinutes () -
getSeconds () -
getTime () -
getYear () -
setDate () -
parse () -
setHours () -
setMinutes () -
setMonth () -
setSeconds () -
toString () -

والمثال يوضح بعض استخداماته لطبع تاريخ اليوم في بداية أية صفحة ويب :-

```
< html >  
< head >  
< script language = " JavaScript " >  
function showHeader ( ) {  
theDate = Date ( ) . toString ( )  
document.writeln ( " < html > < table width =100%  
border=1 > " + " < tr > < td width= 50% align= left >Date :  
" + theDate + < td > < td align = right > < / td > < / tr >  
+ " < / table > " ) ;  
}
```

```
showHeader ( ) ;
```

```
< / script >
```

```
< / head >
```

```
< / html >
```

المثال الثاني على استخدام التاريخ هو لاصدار الوقت في بداية صفحة الويب بحيث يتم تغييره حسب الثواني والدقائق والساعات :-

```
< HTML >
```

```
< HEAD
```

```
< TITLE > Show Time JavaScript Program < / TITLE >
```

```
Time :
```

```
< FORM NAME = " Temps1 " >
```

```
< INPUT TYPE =" text" NAME="heure" SIZE="12"> .
```

```
< / FORM>
```

```
< SCRIPT LANGUAGE = " JavaScript " >
```

```
var dd , delai ;
```

```
function debutTemps ( delai ){
```

```
var hhmmss = " ", min , sec ;
```

```
delai = delai ;
```

```
adate = new Date ( )
```

```
hhmmss += adate . getHours ( ) ;
```

```
min = adate . getMinutes ( ) ;
```

```
if ( min < 10 ) hhmmss += " : 0 " + min ;
```

```
else hhmmss += " : " + min ;
```

```
sec = adate . getSeconds ( ) ;
```

```
if ( sec < 10 ) hhmmss += " : 0 " + sec ;
```

```
else hhmmss += " : " + sec ;
```

```
hhmmss = " " + hhmmss ;
```

```
document. Temps1 . heure.value = hhmmss ;
```

```
dd = setTimeout ( " debutTemps ( delai ) " ,  
delai ) ;}
```

```

</SCRIPT>
</HEAD>
<BODY BGCOLOR = # a3cddd
    VLINK = # 000080
    TOPMARGIN = 100
    LEFTMARGIN = 250
    onLoad = " debutTemps ( 1000 ) "
    onUnload = " clearTimeout ( dd ) " >
</BODY>
</HTML>

```

ب - المتغير القياسي للعمليات الحسابية (Math) :-
 المتغير القياسي للعمليات الحسابية يحتوي على الصفات المرافقة التالية :-

- الرفع (E)
- اللوغاريتم الطبيعي (LN 10)
- اللوغاريتم (LN 2)
- العدد (PI)
- الجذر التربيعي (SQRT 2)
- الجذر للقوة نصف (SQRT 1-2)

وهناك العديد من الاجراءات المبينة عليه ننكر منها التالي :-

cos () - atan () - asin () - acos () - abc () -
 exp () - log () - max () - min () - pow ()
 -round () - sin () - sqrt () - tan ()

ومن امثلة استخدام هذا المتغير القياسي المثاليين التاليين :-

- 1- X = Math.pow (2 , 12) ; يقوم برفع 2 إلى القوة 12
- 2- Pi = Math.PI ; ويقوم بأخذ قيمة باي

ج - المتغير القياسي للخيوط الرمزية (string) :-
والمتغير القياسي للخيوط الرمزية له صفة واحدة وهي الطول (length) وله
كذلك كل من الاجراءات المرافقة التالية :-

anchor () -
big () -
bold () -
charAt () -
fixed () -
fontcolor () -
fontsize () -
indexOf () -
italics () -
link () -
small () -
sub () -
substring () -
sup () -
toLowerCase () -
toUpperCase () -

ومن امثلة استخدامها ، فلو فرضنا أن خيطا رمزيا تم تصريحه كالتالي :-

```
var Name = " Java " ;
```

ولذلك فعند استدعاء Name.length فإن الجواب سيكون 4 لأن طول الخيط
الرمزي هو أربعة حروف وكذلك فعند استدعاء :

```
result = Name.toUpperCase ( ) ;
```

فإنه سيتم تحويل الخيط الرمزي إلى حروف كبيرة .

4-2 - بعض الامثلة على استخدامات نصوص جافا :-

نستعرض هنا بعض برامج نصوص جافا التي تعطينا خبرة في بناء المزيد
منها:-

أ - برنامج لقراءة قيم ثم مضاعفة هذه القيم وإعادة طباعتها :-

```
< html >
< head >
< title > Input Output in Java Script < / title >
< script language = " JavaScript " >
    function readx ( form ) {
        form . result . value = form . giveme. value * 2
    }
< / script >
< / head >
< body >
< title > JavaScript with Input Output < / title >
< hr >
< p >
< form method = post >
    Give Me Your Number and I Will Double it :
< input type=text name=giveme size=30
    onchange= "readx (this.form ) " >
< input type=button value=PUSHME
    onclick="readx(this.form)">
    < / p >
    < p >
    < input type=text name =result size= 3 >
    < /p >
< / form >
< hr >
< / body >
< / html >
```


ب - برنامج لكتابة ارقام غير مرتبة مع كيسة لترتيبهم :-

```
< HTML >
< HEAD >
< TITLE > Sort A Column < / TITLE >
< BODY >
< center > < h1 > WEB SORTING < / h1 > < / center >
< SCRIPT LANGUAGE = " LiveScript " >
function exchange ( i , form ) {
document.forms[6].check.value=document.forms[i].check.value;
document.forms[i].check.value=document.forms[i+1].check. value ;
document.forms[i+1].check.value=document.forms[6].check.value ; }
function checkSort ( form ){
  for ( var j = 0 ; j < 5 ; j ++ ){
    for ( var i = 0 i < 5 ; i ++ ){
      var l = 1 * i + 1 ;
      if ( 1 * document.forms[i].check.value > 1 * document.
        forms [i + 1]. check.value ){
        exchange ( i , form ) ;}
    }
  }
}
< / SCRIPT >
< / HEAD >
< FONT SIZE = 3 >
< CENTER >
< TABLE border = 2 >
< FORM method = POST >
  < TR >
    < TD >
      < DIV ALIGN = CENTER >
```

```

        < INPUT TYPE="button" VALUE="SORT "
                onClick=checkSort( this.form ) >
    < / DIV >
< / TD >
< / TR >
< TR >
< TD >
    < INPUT TYPE=TEXT NAME=check SIZE=6 VALUE=3 >
< / TD >
< / TR >
< / FORM >
< FORM method = POST >
    < TR >
    < TD >
        < INPUT TYPE= TEXT NAME=check SIZE=6 VALUE=7 >
    < / TD >
< / TR >
< / FORM >
< FORM method = POST >
    < TR >
    < TD >
        < INPUT TYPE=TEXT NAME=check SIZE=6 VALUE=1 >
    < / TD >
< / TR >
< / FORM >
< FORM method = POST >
    < TR >
    < TD >
        < INPUT TYPE=TEXT NAME=check SIZE=6 VALUE= 33 >
    < / TD >
< / TR >

```

```

</FORM>
<FORM method = POST>
  <TR>
    <INPUT TYPE=TEXT NAME=check SIZE=6 VALUE = 22>
  </TD>
</TR>
</FORM>
<FORM method = POST>
  <TD>
    <INPUT TYPE=TEXT NAME=check SIZE=6 VALUE= 11>
  </TD>
</TR>
</FORM>
</TABLE>
</CENTER>
<FORM method = POST>
  <TR>
    <TD>
      <INPUT TYPE= HIDDEN NAME=check SIZE=6>
    </TD>
  </TR>
</FORM>
</FORM>
</BODY>
</HTML>

```

ج - برنامج لإصدار لوحة إعلان على صفحة الويب :-

```
< HTML >
< HEAD >
< TITLE > BANNER WEB PAGE < / TITLE >
< FORM NAME = " formBanl " >
  < INPUT TYPE="text " NAME=" Fbannierel " SIZE="40"> < BR>
< / FORM >
< SCRIPT LANGUAGE = " JavaScript " >
  var posBanl = 0 , banl , delaiBanl , msgBanl ;
  function bannierel ( delai ) {
    delaiBanl = delai ;
    if ( postBanl >= msgBanl.length )
      posBanl = 0 ;
    else if ( posBanl == 0 ) {
      msgBanl = '      ' + msgBanl ;
      while ( msgBanl.length < 128 )
        msgBanl += '      ' + msgBanl ;
    }

document.formBanl.Fbannierel.value=msgBanl.substring(posBanl,
                                                    posBanl + msgBanl . length ) ;

    posBanl ++ ;
    banl = setTimeout ("bannierel (delaiBanl) ", delai ) ;
  }
< / SCRIPT >
< / HEAD >
< BODY BGCOLOR = # FFFFFFFF VAINK = # 000080
  TOPMARGIN=80 LEFTMARGIN=80 onLoad = "msgBanl =
    ' Please this is my HTML Bannar ' ;bannierel(100);
    "clearTimeout(banl)" >
< FORM NAME = "Temps 2" >
```

```

</ FORM >
< CENTER > < IMG SRC = " Backgd03 . jpg " BORDER = 0
                ALT = " MY IMAGE " > < br >

</ BODY >
</ HTML >

```

د - برنامج لإخراج نافذة أخبار في بداية صفحة الويب :-

```

< HTML >
< HEAD >
< TITLE > NEWS PROGRAM < / TITLE >
< FORM NAME = " formnouv " >
    < TEXTAREA NAME ="multil" WARP=PHYSICAL
        COLS="40 "" ROWS =" 8 " > < / TEXTAREA >
< / FORM >

< SCRIPT LANGUAGE = " JavaScript " >
var pos1 = 0 , pos2 = 0 , nouvDelai , Fin 2, MsgN ;
function TexteMultiligne (nouvdelai1 ) {
    nouvDelai = nouvDelai1;
    if ( pos1 >= MsgN.length ) {
        document. formnouv . multi1.value = ' ' ;
        pos1 = 0 ;
        pos2 = 0 ;
    }
    else if ( MsgN . substring ( pos1 - 2 , pos1 - 1 ) == ' . ' ) {
        document. formnouv. multi1. value = ' ' ;
        pos2 = pos1 - 1 ;
        pos1 ++ ;
    }
    else {
        document.formnouv.multi1.value=MsgN.substring(pos1, pos2) ;
    }
}

```

```

    pos1 ++ ;
}
Fin2=setTimeout("TexteMultiligne( nouvDelai )" , nouvDelai1 ) ;
}
</ SCRIPT >
</ HEAD >
< BODY BGCOLOR = # abcabc VAINK = # 000080
TOPMARGIN=0 LEFTMARGIN= 220 onLoad="MsgN =
    'This is a news box: No News is Good News . ' ;
    TexteMultiligne ( 100 ) ; " >
< CENTER > < IMG SRC = " Backgd03 . jpg " BORDER
= 0
    ALT = " MY SECOND IMAGE" > < br >
</ BODY >
</ HTML >

```

الفصل الثالث

ثلاثة طرق للتفاعل مع جافا

الفصل الثالث

ثلاثة طرق للتفاعل مع جافا

3-1- التفاعل عن طريق الادخال والاخراج :-

أن لغة جافا يمكن تصورها في البداية على أنها لغة برمجة اعتيادية مثل لغة C أو Pascal وبالتأكيد فإن هذه اللغة تحتوي نفس التركيبة من العبارات البرمجية وبالذات فأت هذه اللغة يمكنها القراءة والكتابة بنفس الطريقة التي تقوم بها اللغات الأخرى ، ولنأخذ مثلاً يوضح عملية قراءة أي عدد من الحروف (characters) من لوحة المفاتيح ويقوم بطباعة عدد هذه الحروف ويخرج عندما يكون الحرف الأخير الذي نطبعه هو 1- :-

```
class Count {  
    public static void main ( String [ ] args )  
        throws java . io . IOException  
    {  
        int count = 0 ;  
        while ( System . in . read ( ) != - 1 )  
            count ++ ;  
        System . out . println ( " Input has " + count + " chars . " ) ;  
    }  
}
```

ويلاحظ أن عملية القراءة تمت من خلال استخدام اجراءات مبنية وهي **System . in . read ()** للقراءة و **System . out . println** للطباعة ، ويلاحظ أن البرنامج يستخدم **IOException** وهو اجراء للقيام بتأشير الاخطاء الممكن حدوثها ولكن طريقته غير مكتملة في هذا المثال حيث لا يوجد مكان لاستلام الاخطاء واعطاء

عبارات مقابلة ، ولأخذ مثال اوضح يقوم بقراءة (string) وبطاعته ونأشير حاله خطأ عندما يكون الحيط الرمزي خالياً :-

```
import java . io . * ;
class readw {
public static void main ( String [ ] args ) throws IOException{
DataInputStream in = new DataInputStream ( System . in ) ;
String tt ;
System . out . println ( " What is your string = " ) ;
try {
tt = in . readLine ( ) ;
System.out.println("The String given was "+ String.valueOf( tt ) ;
} catch ( IOException e ) { tt = " ZERO STRING " ; }
}
}
```

أما حول كيفية التعامل مع القيم العددية فإن اجراء System . in . readln يقوم

بقراءة العدد كخييط رمزي وهذا يتطلب تحويله إلى عدد ، مثلاً عدد حقيقي Double . البرنامج التالي يقوم بقراءة وزن شخص على الارض ويقوم بتحويله إلى وزن على سطح القمر ومن ثم طباعة الوزن الجديد على القمر :-

```
import java . io . * ;
class readw {
public static void main ( String [ ] args ) throws IOException {
DataInputStream in = new DataInputStream ( System . in ) ;
String tt ; double ntt ;
System . out . println ( " What is your weight on earth = " ) ;
try {
tt = in . readLine ( ) ;
ntt = Double . valueOf ( tt ) . doubleValue ( ) ;
System . out . println ( " Your weight on moon is = " ntt * . 166 ) ;
} catch ( IOException e ) { tt = " ZERO STRING " ; }
}
}
```

ويلاحظ أن طبع الخطأ يتم اقتناصه ويمكن طبع إشارة خطأ من قبل المستخدم مثل **Zero** String " أو يمكن للبرنامج طبع الخطأ القياسي الذي يؤشره النظام مثل البرنامج التالي الذي يقوم بقراءة ملف معين باسم **farrago . txt** وطبعه على ملف آخر مثل **outagain . txt**:

```
import java . io . * ;
class FileStreamsTest {
    public static void main ( String [ ] args ) {
        try {
            File inputFile = new File ( " farrago . txt " ) ;
            File outputFile = new File ( " outagain . txt " ) ;
            FileInputStream fis = new FileInputStream ( inputFile ) ;
            FileOutputStream fos= new FileOutputStream ( outputFile ) ;
            int c ;
            while ( ( c = fis . read ( ) ) != - 1 ) {
                fos .write ( c ) ;
            }
            fis .close ( ) ;
            fos .close ( ) ;
        } catch ( FileNotFoundException e ) {
            System . err . println ( " FileStreamsTest : " + e ) ;
        } catch ( IOException e ) {
            System . err . println ( " FileStreamsTest : " + e ) ;
        }
    }
}
```

ويلاحظ أن الخطأ القياسي يتم من خلال استخدام **System . err . println** ومن الجدير بالذكر أن هناك حالات عديدة لاستخدام عبارة الطباعة المبنية **System**

err . println ، والمثال التالي يوضح استخدامها مع عمليات حسابية :-

```
1: class ArithmeticTest {
2:     public static void main ( string [ ] args ) {
3:         short x = 6 ;
4:         int y = 4 ;
5:         float a = 12.5f ;
6:         float b = 7f
7:
8:         System . out . println ( " x is " + x + " , y is " + y ) ;
9:         System . out . println ( " x + y - " + ( x + y ) ) ;
}
```

```

10: System.out.println("x - y = " + (x - y));
11: System.out.println("x / y = " + (x / y));
12: System.out.println("x % y = " + (x % y));
14: System.out.println("a is " + a + ", b is " + b);
15: System.out.println("a / b " + (a / b));
16: } }

```

وتكون مخرجات هذا البرنامج كالتالي :-

```

x is 6 , y is 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
a is 12.5 , b is 7
a / = 1.78571

```

والمثال الثاني يوضح استخدام **System.out.println** مع الحيلولة الرمزية :-

```

1: class TestString {
3: public static void main ( string [] args ) {
4: String str = " Now is the winter of our discontent " ;
6: System.out.println ( " The string is : " + str );
7: System.out.println ( " length of this string : "+ str.length() );
9: System.out.println ( "The char at position 5 : "+ str.charAt(5) );
11: System.out.println ( "The substr 11-18: "+ str.substring ( 11,18 ) );
13: System.out.println ( "The index of the char d : "+ str.indexOf ( 'd' ) );
15: System.out.print ( "The index of the beginning of the " );
16: System.out.println ( "substr \"winter \" : "+ str.indexOf ( "winter" ) );
18: System.out.println ( "The str in upper case : "+ str.toUpperCase() );
20: } }

```

وتكون مخرجات هذا البرنامج كالتالي :-

The string is : Now is the winter of our discontent

Length of this string : 35

The character at position 5 : s

The substring from position 11 to 18 : winter

The index of the character d : 25

The index of the beginning of the substring " winter " : 11

The str in upper case: NOW IS THE WINTER OF OUR DISCONTENT

وبالاحظ أن معظم برنامج جافا تحتوي على إجراء رئيسي (main method)

ويحتوي في داخله عدد من المدخلات arguments ويستخدم لذلك الرمز [] arg والبرنامج التالي يوضح كيف يمكن عند تشغيل البرنامج تمرير عدد من المدخلات الى البرنامج:-

```
1: class EchoArgs {  
2:     public static void main ( String args [ ] ) {  
3:         for ( int i=0 ; i < args . length ; i ++ ) {  
4:             System.out.println ( "Argument"+i + " : " + args [i] ) ;  
5:         }  
6:     }  
7: }
```

والتالي طريقتين لتشغيل البرنامج بإعطائه مدخلات مختلفة :-

java EchoArgs 1 2 3 jump



Arguments 0 : 1
Arguments 1 : 2
Arguments 2 : 3
Arguments 3 : jump

java EchoArgs " foo bar " zap twaddle 5



Arguments 0 : foo bar
Arguments 1 : zap
Arguments 2 : twaddle
Arguments 3 : 5

والبرنامج أعلاه يقوم بطبع أي عدد من المدخلات نقوم بإعطائها له .
ولا تقتصر عملية الإدخال والإخراج بين المستفيد والبرنامج بل تعداها إلى وجود تفاعل في الإدخال والإخراج بين إجراءات البرنامج نفسه والمثال التالي يوضح عملية تبادل المدخلات والمخرجات بين إجرائيين داخل البرنامج الواحد الذي نطلق عليه

PassByReference الذي يوجد فيه اجراء تحويل عدد الواحدات إلى أصفار (**OnetoZero**) والاجراء الثاني، الاجراء الرئيسي (**main**) ، ويقوم البرنامج بقراءة قيمة مصفوفة (**array**) داخل الاجراء الرئيسي ومن ثم طبع قيمة هذه المصفوفة وبالتالي إرسال هذه المصفوفة إلى الاجراء **OnetoZero** التي نحسب فيها عدد الواحدات وكذلك نغير في المصفوفة كل الواحدات إلى أصفار واعادة هذه المصفوفة بعد تغييرها وعدد الواحدات من خلال المتغير (**count**) إلى الاجراء الرئيسي الذي يقوم بدوره بطباعة المصفوفة وعدد الواحدات التي كانت أصلاً فيها ، ويفيدنا هذا المثال في التعرف على كيفية التعاطب بين الاجراءات الداخلية المتوفرة داخل برنامج واحد (**class**) .

Type

```

1: class PassByReference {
2:     int OnetoZero ( int arg [ ] ) {
3:         int count = 0 ;
4:         for ( int i = 0 ; i < arg . length ; i ++ ) {
5:             if ( arg [ i ] == 1 ) {
6:                 count ++ ;
7:                 arg [ i ] = 0 ;
8:             }
9:         }
10:    }
11:    return count ;
12: }
13: }

1: public static void main ( String arg [ ] ) {
2:     int arr [ ] = { 1 , 3 , 4 , 5 , 1 , 1 , 7 } ;
3:     PassByReference test = new PassByReference ( ) ;
4:     int numOnes ;
5:     System . out . print ( " Value of the array : [ " ) ;
6:     for ( int i = 0 ; i < arr . length ; i ++ ) {
7:         System . out . print ( arr [ i ] + " " ) ;
8:     }
9:     System . out . println ( " ] " ) ;

```

```

12: numOnes = test . OnetoZero ( arr );
13: System . out . println ( " Number of Ones = " + numOnes );
14: System . out . print ( " New value of the array ; [ " );
15: for ( int i = 0 ; i < arr . length ; i ++ ) {
16:     System . out . print ( arr [ i ] + " " );
17: }
18: System . out . println ( " ] " );}

```

ونتائج البرنامج أعلاه هي كالتالي :-



```

Value of the array : [ 1 3 4 5 1 1 7 ]
Number of Ones = 3
New value of the array : [ 0 3 4 5 0 0 7 ]

```

وكمثال أفضل على مخاطب الاجراءات الداخلية بين بعضها الآخر نأخذ المثال التالي

الذي يقوم بأخذ صورة من المتغيرات وإعطائها قيم وطبعها مرة ثم تغيير صورة المتغيرات وطبعها

مرة اخرى :-

```

class Person {
    String name ;
    int age ;
    Person ( String n , int a ) {
        name = n ;
        age = a ;
    }
    void printPerson ( ) {
        System . out . print ( " Hi , my name is " + name ) ;
        System.out.println("I am " + age + " years old") ;
    }
    public static void main ( String args [ ] ) {
        Person p ;
        p = new Person ( " Laura " , 20 ) ;
        p . printPerson ( ) ;
        System . out . println ( " ..... " ) ;
        p = new Person ( " Tommy " , 3 ) ;
        p . printPerson ( ) ;
        System . out . println ( " ..... " ) ;
    }
}

```

ونائج البرنامج كالتالي :-

Output

Hi , my name is Laura . I am 20 years old .

.....

Hi , my name is Tommy . I am 3 years old .

.....

ويلاحظ أن كل نسخة من المتغير **person** أعطيت له قيم وتم طبعتها تباعاً . وكمثال

آخر نأخذ المتغير القياسي **Data** ونعطيه كل مرة قيم معينة ونقوم بطباعتها :-

```
1: import java , util , Data ;
3: class CreateDates {
5:     public static void main ( String args [ ] ) {
6:         Data d 1 , d 2 , d 3 ;
7:
8:         d 1 = new Data ( ) ;
9:         System . out . println ( " Data 1 : " + d 1 ) ;
10:
11:         d 2 = new Data ( 71 , 7 , 1 , 7 , 30 ) ;
12:         System . out . println ( " Data 2 : " + d 2 ) ;
13:
14:         d3 = new Data ( " April 3 1993 3 : 24 PM " ) ;
15:         System . out . println ( " Data 3 : " + d 3 ) ;
16:     }
17: }
```

Output

ونائج هذا البرنامج تكون كالتالي :-

Date 1 : Sun Nov 26 19 : 10 : 56 PST 1995

Date 2 : Sun Aug 01 07 : 30 : 00 PDT 1971

Date 3 : Sat Apr 03 15 : 24 : 00 PST 1993

3-2- التفاعل عن طريق النوافذ (AWT) :-

إن طريقة التفاعل مع المستفيد في هذا النوع من البرمجيات تعتمد على استخدام الاجراءات المبنية في حزمة AWT (Abstract Windows Toolkit) التي تمكن المستفيد من اجراء العمليات التالية :-

- رسومات (Graphics) .
- فتح نوافذ للتفاعل (Windows) وقوائم اختيارات (Menus) .
- عمل وادارة تسلسل الاعمال (Layout Manger) .
- ادارة الاعمال وتنفيذها بشكل متوازي (Event Handling) .
- معالجة الصور (Image Manipulation) .

والفضل طرق التفاعل تكون باستخدام النوافذ وهذا يتطلب فتح النافذة داخل ايطار (Frame) ولذلك فعند تعريف أي برنامج يقوم بالتفاعل بواسطة النوافذ تكون صيغة تعريفه كالتالي :-

```
public classname extends Frame {  
    classname ( String title ) {  
        Super ( title )  
        More constructive actions  
    }  
    other methods  
}
```

ويحتاج تعريف الايطار داخل الاجراء الرئيسي إلى العبارة التالية :-

```
Frame f = new classname ( "title" )  
init ( ) ; show ( ) ;
```

ويجب التنبيه إلى أن تصريح الايطار يحتاج إلى اجراء لتحديد حجمه نطلق عليه بصورة طبيعية اجراء التهيئة (init) والذي يكون فيه عبارة تحديد العرض والطول للايطار (resize width , hight) وكذلك إلى اجراء للرسم داخل هذا الايطار يمكن أن نطلق عليه paint وبالتأكيد نحتاج إلى اجراء للخروج من هذه النافذة نطلق عليه handleEvent .

والمثال التالي يقوم بطبع عبارة Helloooooooooo في داخل ايطار بعرض 200 وبطول 60 .

```
import java . awt . Graphics ;
import java . awt . Frame ;
class Ring extends Frame {
    Ring ( String s ) {
        super ( s ) ;
    }
    void init ( ) {
        resize ( 200 , 60 ) ;
    }
    public void paint ( Graphics g ) {
        g . drawString ( " Helloooooooooo " , 30 , 25 ) ;
    }
    public static void main ( String args [ ] ) {
        System.out.println( " entering main ..... " ) ;
        Ring f = new Ring ( " Ring Frame " ) ;
        f . init ( ) ;
        f . show ( ) ;
        System . out . println ( "leaving main ..... " ) ;
    }
    public boolean handleEvent ( Event e ) {
        if ( e . id == Event . WINDOW _ DESTROY )
            System . exit ( 0 ) ;
        return super . handleEvent ( e ) ;
    }
}
```

وبلاحظ أن عملية اغلاق الايطار تحتاج إلى مساعدة ادارة العمليات (Event Handler) وهذه الاداة تحتاجها في العديد من العمليات الاخرى ولأخذ مثال يوضح تدخّل ادارة العمليات هو عملية محاكاة اشارة ضوئية فيها كهستين إحداها لطلب المارة للعبور والثانية لالغاء طلب العبور من المارة . وطلب المارة للعبور خلال عمل الاشارة الضوئية يتطلب ادارة خاصة يوضحها البرنامج التالي :-

```

import java . awt . * ;
import java . io . * ;
public class traffic extends Frame {
public traffic ( ) {
    setTitle ( " traffic lights " );
    Panel title = new Panel ( );
    title . add ( new Label ( " traffic light problem " ) );
    add ( " North " , title );
    lights = new LightsCanvas ( );
    add ( " Center " , lights );
    Panel buttons = new Panel ( );
    buttons . add ( new Button ( " Walk " ) );
    buttons . add ( new Button ( " Cancel " ) );
    Choice c = new Choice ( );
    c . addItem ( " Red " );
    c . addItem ( " Yellow " );
    c . addItem ( " Green " );
    c . addItem ( " Walk " );
    buttons . add ( c );
    buttons . add ( new Label ( " Duration " ) );
    Text Field duration = new TextField ( " " , 3 );
    duration . setEditable ( false );
    buttons . add ( duration );
    add ( " South " , buttons );
}
public boolean handleEvent ( Event evt ) {
    If ( evt . id == Event . WINDOW_DESTROY ) System . exit ( 0 );
    return super . handleEvent ( evt );
}
public boolean action ( Event evt , object arg ) {
    boolean reply = true ;
    if ( arg . equals ( " Cancel " ) ) {
        this . hide ( );
        this . dispose ( );
        System . exit ( 0 );
    } else
        if ( arg . equals ( " Walk " ) ) {

```

```

Panel b = new Panel ();
b.add ( new Button ( " jinan " ));
TextField d = new TextField ( "   " , 12 );
d.setEditable ( true );
d.setText ( " faidhi " );
b.add ( d );
add ( " East " , b );
}
repaint ();
return false ; }

public static void main ( String [] args ) {
    Frame f = new traffic ();
    f.resize ( 300 , 200 );
    f.show ();
}

private LightsCanvas lights ;
}

class LightsCanvas extends Canvas {
public void paint ( Graphics g ) {
    g.drawOval ( 97 , 10 , 30 , 68 );
    g.setColor ( Color . red );
    g.fillOval ( 105 , 15 , 15 , 15 );
    g.setColor ( Color . yellow );
    g.fillOval ( 105 , 35 , 15 , 15 );
    g.setColor ( Color . green );
    g.fillOval ( 105 , 55 , 15 , 15 );
    g.fillOval ( 105 , 85 , 15 , 15 );
    g.setColor ( Color . black );
    g.drawString ( " Red " , 15 , 28 )
    g.drawString ( " Yellow " , 15 , 48 )
    g.drawString ( " Green " , 15 , 68 )
    g.drawString ( " Walk " , 15 , 98 )
}
}
}

```

ويمكن تطوير هذا البرنامج لكي يقوم بإدارة الإشارة الضوئية بشكل متكامل.

والسؤال هل نستطيع القراءة داخل النوافذ ؟ والجواب نعم ولتوضيح ذلك نأخذ المثال التالي الذي يقوم بقراءة وزن الشخص على الارض وطبع وزنه على القمر ، وبلاحظ أن عملية القراءة تتم بواسطة (`getText`) كالتالي :-

```
import java . awt . *
public class PlanetaryScale extends Panel {
    Label label ;
    Textfield textField ;
    Astronaut armstrong ;
    void calculatew () {
        armstrong = new Astronaut ( get EarthWeight () ) ;
        showMoonWeight ( armstrong . moonWeight () ) ;
    }
    double getEarthWeight () {
        double wt ;
        try {
            wt=Double.valueOf( textField . getText () ).doubleValue () ;
        } catch ( java . lang . NumberFormatException e ) {
            wt = 0 . 0 ;
        }
    }
    void . showMoonWeight ( double f ) { ;
    label . setText ( " Your moon weight is " + String . valueOf ( f ) ) ;
    public void init () {
        resize ( 400 , 60 ) ;
        textField = new TextField ( 6 ) ;
        add ( textField ) ;
        label = new label ( " Enter your earth weight = " ) ;
        add ( label ) ;
        armstrong = new Astronaut ( 0 , 0 ) ;
    }
    public boolean handleEvent ( Event e ) {
        if ( e . target instanceof TextField && e . id ==
            Event . ACTION _ EVENT ) {
            calculatew () ;
            return true ;
        }
        return false ;
    }
}
```

```

public static void main ( String args [ ] ) {
    PlantaryScale ps = new PlantaryScale ( );
    ps . init ( );
    Frame f = new Frame ( " PlanetaryScale " );
    f . resize ( 400 , 60 );
    f . add ( " Center " , ps );
}
}

```

ويلاحظ أن عملية القراءة تكون داخل الاطار باستخدام عبارة (`textField . getText ()`) ، والكتابة تكون باستخدام عبارة اخرى هي :

```

lable . setText ( " Things Name " + String . valueOf ( f ) ) ;

```

ويلاحظ كذلك استخدام عبارة `add` التي تقوم باضافة قيم جديدة إلى النافذة ، وأخيراً نلاحظ استخدام `handleEvent` التي تقوم برصد أن المستخدم قد ادخل قيمة داخل الحقل `textField` وعندها يعطي الاشارة لتنفيذ العمليات التالية ، وبالتأكيد نحن نحتاج إلى اجراءات تحديد بعد الاطار (`init ()`) وكذلك اجراء للخروج من الاطار وهي الاجراءات التقليدية التي ناقشناها في المثال السابق . إن الملاحظة العامة التي يجب ذكرها هنا أن برمجة النوافذ تحتاج إلى العديد من الاجراءات التي يجب أن يقوم بها المبرمج مثل اجراء الخروج واجراء تنظيم العمليات وغيرها .. مما يؤكد صعوبة العمليات البرمجية حيث لا يوجد ما هو معروف مسبقاً للمبرمج للعمليات الاساسية التي يمكن أن تتكرر في كل نافذة ، ولذلك فإن طريقة التفاعل باستخدام الابلتات التي سنناقشها في الفقرة القادمة هي من النوع الذي يعطي النوافذ عمليات اساسية معرفة له مسبقاً .

3-3- التفاعل بالاعتماد على الابلت (Applets) :-

إن الابلت هو برنامج مكتوب بلغة جافا ولكنه يعمل مع معالج صفحات الويب (`Browser`) (ولذلك فهو يمكن أن يوزع على عموم الشبكة ، وهناك فوائد لاستخدام الابلت هو أن عملية تنفيذ أي ويب يكون في المكان الذي يتم التنفيذ فيه (`Executed at site needed`)

(وكذلك يمكن للابليت أن يستخدم كل مواصفات لغة جافا المتوفرة ، ولكون الابليت يعمل مع معالج صفحات الويب فهو سوف يخلق نافذة (Window) في داخل صفحة الويب وهذا يعني أنه سوف يستخدم حزمة النوافذ (awt) ، والابليت في الواقع هو برامج لغة ارشادية (HTML) يستدعي في داخله البرنامج المترجم (compiled) لبرنامج جافا ، والبرنامج التالي يوضح طريقة خلق أول أبليت الذي يقوم بطباعة عبارة أهلاً بالعالم داخل نافذة في صفحة ويب :-

```
import java . applet ;
import java . awt . Graphics ;
public class HelloApplet extends Applet {
    public void init ( ) {
        resize ( 200 , 60 ) ;
    }
    public void paint ( Graphics g ) {
        g . drawString ( " Hello , World " , 60 , 25 ) ;
    }
}
```

وبرنامج اللغة الارشادية المرافق هو :-

```
< HTML >
< HEAD >
< TITLE > A Simple Program < / TITLE >
< / HEAD >
< body >
here is the output of my simple program :
<APPLET CODE="HelloApplet.class"width=150 height=25 >
< / APPLET >
< a href = " HelloApplet . java " > The source . < / a >
< / body >
< / html >
```

```

import java . applet . * ;
import java . awt . * ;
public class HelloWorld extends Applet ;
Font f = new Font ( " timesRoman " , font . Bold , 36 ) ;
    public void init ( ) {
        setBackground ( Color , white ) ;
    }
    public void paint ( Graphics g ) {
        g . setFont ( f ) ;
        g . setColor , red ) ;
        g . deawString ( " Wow - Red " , 100 , 26 ) ; } }

```

ويلاحظ أن الابلت لا يحتاج إلى اجراءات مساندة اساسية مثلاً اجراء لخلق النافذة المفصولة داخل صفحة الويب التي كنا نحتاجها عندما نستخدم طريقة النوافذ في التفاعل (awt) وبالتأكيد نحن لن نستخدم المفسر (java) لتنفيذ الابلت وهو ينفذ مباشرة من قبل معالج صفحات الويب (Browser) من خلال تنفيذ برنامج اللغة الارشادية المرافق والذي يستدعي برنامج الابلت الذي قمنا بترجمته من قبل مترجم جافا (javac) ويمكن أن نستخدم الابلت لاصدار نوافذ تحتوي على كلام مكتوب (Text) أو يحتوي على رسومات وصور (Graphics) أو أشكال أخرى أو ممزوجة ببعضها ، ومنحاول أن نعطي بعض الأمثلة على استخدامات الابلت المفردة ، ولنبداً باصدار ابلت يقوم بإخراج نوافذ تحتوي على نصوص مثل المثال أعلاه. ويلاحظ أن هذا الابلت استخدم بعض الاجراءات التي تعمل على النص مثل :-

- 1- كتابة الحرف `g. drawString`
- 2- اختيار شكل الحرف `g. setFont`
- 3- اختيار لون الحرف `g. setColor`

وهذا الابلت يقوم بطبع عبارة `Wow -- Red` على صفحة الويب ، ويمكن أن يستخدم اجراء كتابة الحروف `g. drawString` لطباعة أي قيمة لحيط رمزية موجودة داخل متغير كما في المثال التالي :


```

import java . awt . * ;
import java . applet . Applet ;
public class Applet extends Applet {
    string stringWithValue = " , Hey , I'm a string ! " ;
    font f = new Font ( " TimesRoman " , Font . BOLD , 36 ) ;
    public void init ( ) {
        setBackground ( Color . white ) ;
    }
    public void paint ( Graphics g ) {
        g . setFont ( f ) ;
        g . drawString ( this . stringWithValue , 50 , 150 ) ;
    } }

```

وهناك العديد من الاجراءات المبنية التي تعمل على الخيوط الرمزية:

length () طول الخيط الرمزي

toLowerCase () دالة التحويل إلى الحروف ذات الحجم الصغير

toUpperCase () دالة التحويل إلى الحروف ذات الحجم الكبير

ويوضح المثال التالي استخدام هذه الدوال :-

```

import java . awt . * ;
import java . applet . Applet ;
public class Applet4 extends Applet {
    String stringWithValue = " Hey , I'm a string ! " ;
    Font f = new Font ( " TimesRoman " , Font . Bold , 20 ) ;
    public void init ( ) {
        setBackground ( Color . white ) ;
    }
    public void paint ( Graphics g ) {
        g . setFont ( f ) ;
        g . drawString ( " The string is : " + this . stringWithValue , 50 , 50 ) ;
        g . drawString ( "The string is:" + this . stringWithValue + "\n" , 50 , 75 ) ;
        g . drawString ( "The string's length:" + this . stringWithValue . length ( ) , 50 , 100 ) ;
        g . drawString ( "Lowercase:" + this . stringWithValue . toLowerCase ( ) , 50 , 125 ) ;
        g . drawString ( "Upper case:" + this . stringWithValue . toUpperCase ( ) , 50 , 150 ) ;
    } }

```

أما طريقة اخراج رسومات بسيطة من خلال الابلت فتم باستخدام اجراءات أهمها التالي :-

- | | |
|--------------------------|----------------------------|
| drawRoundRect () | 1- رسم مستطيل ببعدين |
| draw3DRect () | 2- رسم مستطيل بثلاثة ابعاد |
| drawOval () | 3- رسم شكل بيضوي |
| drawPolygon () | 4- رسم مضلع |
| drawString () | 5- رسم خيط رمزي |
| fillRect () | 6- ملئ مستطيل |
| fillRoundRect () | 7- ملئ مستطيل ثمانية مدورة |
| fill3DRect () | 8- ملئ مستطيل بثلاثة ابعاد |
| fillOval () | 9- ملئ شكل بيضوي |
| fillPolygon () | 10- ملئ مضلع |
| getColor () | 11- الحرف على لون الشكل |
| setColor () | 12- تثبيت لون الشكل |
| getFont () | 13- الحصول على شكل الحرف |
| setFont () | 14- تثبيت شكل الحرف |
| Graphics () | 15- تحويل إلى الرسومات |
| create () | 16- خلق شكل |
| dispose () | 17- مسح شكل |
| clearRect () | 18- ازالة مستطيل |
| clipRect () | 19- خزن مستطيل |
| drawLine () | 20- رسم خط |
| drawRect () | 21- رسم مستطيل |

ولتوضيح استخدام هذه الدوال (methods) نأخذ الامثلة التالية :-

أبليت لرسم خط مستقيم:

```
// Draw a line
import java . awt . * ;
import java . applet . * ;
public class Line extends java . applet . Applet {
    public void paint ( Graphics g ) {
        g . drawLine ( 50 , 50 , 100 , 150 ) ;
    }

    public void start ( ) {
        repaint ( ) ;
    }
}
```

أبليت لرسم مستطيل:

```
// Draw a Rectangle
import java . awt . * ;
import java . applet . * ;
public class Rectangle extends java . applet . Applet {
    public void paint ( Graphics g ) {
        g . drawRect ( 50 , 50 , 100 , 150 ) ;
    }

    public void start ( ) {
        repaint ( ) ;
    }
}
```

البيات لرسم أشكال مملوءة:

```
// Draw Filled Shapes
import java . awt . * ;
import java . applet . * ;
public class fills extends java . applet . Applet {
    public void paint ( Graphics g ) {
        g . setColor ( Color . blue ) ;
        g . fillRect ( 50 , 50 , 100 , 150 ) ;
        g . setColor ( Color . Red ) ;
        g . fillOval ( 200 , 50 , 100 , 150 ) ;
    }
    public void start ( ) {
        repaint ( ) {
    }
}
}
```

البيات لرسم مضلعات مملوءة:

```
// Draw a Polygon and Filled Polygon
import java . awt . * ;
import java . applet . * ;
public class Poly extends java . applet . Applet {
    int x [ ] = { 100 , 200 , 250 , 50 , 100 } ;
    int y [ ] = { 50 , 50 , 200 , 200 , 50 } ;
    int a [ ] = { 300 , 350 , 400 , 300 } ;
    int b [ ] = { 200 , 50 , 200 , 200 } ;
    public void paint ( Graphics g ) {
        g . drawPolygon ( x , y , 5 ) ;
        g . setColor ( Color . blue ) ;
        g . fillPolygon ( a , b , 4 ) ;
    }
    public void start ( ) {
        repaint ( ) ;
    }
}
```

إن الأبلت يتعامل مع الأرقام بنفس الطريقة التي يتعامل مع الحووط الرمزية ولطابعها يستخدم نفس الاجراء `g.drawString` والمثال التالي يطبع لنا أكبر قيمة (`max`) لكل نوع من أنواع الأرقام `integer` , `Short integer` , `long integer` , `float` , `double` . (`integer` , `Byte`

```
import java . awt . * ;
import java . applet . Applet ;
public class Applet extends Applet {
    byte myByte = Byte . MAX_VALUE ;
    short myShort = short . MAX_VALUE ;
    int myInt = Integer . MAX_VALUE ;
    long myLong = Long . MAX_VALUE ;
    float myFloat = Float . MAX_VALUE ;
    double myDouble = Double . MAX_VALUE ;
    Font f = new Font ( " TimesRoman " , Font . BOLD , 16 ) ;
    public void init ( ) {
        setBackground ( Color . white ) ;
    }
    public void paint ( Graphics g ) {
        g . setFont ( f ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myByte ,10,20 ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myShort ,10,40 ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myInt , 10 , 60 ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myLong , 10 , 80 ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myFloat ,10,100 ) ;
        g . drawString ( "The maximum value of a byte is : "+ this . myDouble ,10,120 ) ;
    }
}
```

إن هذا الفصل حاول استعراض بعض البرمجيات البسيطة التي يتفاعل المستخدم بواسطتها مع برمجيات جافا ، والقارئ الآن قد تمهي للدراسة معمقة لتركيب برمجيات جافا وهو موضوع الفصل الرابع ، ويلاحظ أن اشمل طرق التفاعل تلك التي تستخدم الأبلت حيث

يمكن بواسطتها طباعة رسومات ونصوص وأرقام وهي شاملة لكل الأغراض وبالذات في مجال بناء صفحات الويب ، ولذلك سنركز في الفصول القادمة على استخدام التفاعل بطريقة الابلت ، إن استعراضنا لطرق التفاعل أثبت بطريقة غير مباشرة بأن لغة جافا لغة يمكن تصورها بالتقليدية حيث عندما تتفاعل بطريقة الإدخال والإخراج فأن ذلك التفاعل يشبه إلى حد بعيد لغات مثل لغة سي وعندما تستخدم التوافق فأن ذلك يشبه إلى حد بعيد لغات مثل لغة بيرسك المرنية أما طريقة التفاعل بالابلت فهي أشمل من الطريقتين وهي لأغراض النشر في صفحات شبكة الانترنت وكذلك التفاعل مع بقية مستطدي الشبكة بمختلف الوسائل .

الفصل الرابع

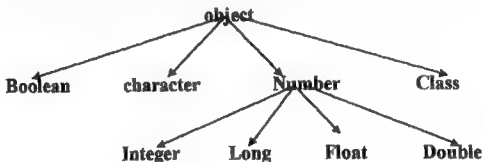
تركيبه برامجيات جافا

الفصل الرابع

تركيبة برامجيات جافا

4-1- مكونات تركيبة جافا الاساسية :-

إن لغة جافا بصورة اساسية هي عبارة عن مكونات حية في الذاكرة (objects) .
وفي الواقع أن ، المكونات الحية في الذاكرة هي أنواع كثيرة أهمها موضحة في الشكل التالي :-



وكل هذه المكونات يمكن أن تنفرع إلى أجزاء فرعية كثيرة ، وأهم هذه المكونات الحية في الذاكرة هو المصنفة (class) المبرمجة وهذه المصنفة المبرمجة هي أداة للتفاعل قد تحتوي قوائم أو كسبات أو صور وكتابات ورسومات وغيرها من ادوات التفاعل ، ولذلك فإن المصنفة المبرمجة هو جزء برمجي هام يحتوي على عبارات واجراءات وقطع برمجية كثيرة لاستخراج التفاعل المطلوب ، والمثال التالي يوضح تصريح مصنفة مبرمجة cloud باسم والتي تنادي في داخلها مصنفة مبرمجة اخرى باسم cirus والتي هي تنادي في داخلها اجراء واحد من النوع البنائي (constructor)

```
class cloud {
    public static void main ( String args [ ] ) {
        System . out . println ( " main " ) ;
        cirus c = new cirus ( ) ;
    }
```

```

class cirus {
    static int arr [ ] = new int [ 3 ];
    static { System . out . println ( " static initializer " );
        for ( int i = 0 ; i <= arr . length ; itt )
            arr [ i ] = i ;
    }
    cirus ( ) { System . out . println ( " Constructor " );
        for ( int i = 0 ; i < arr . length ; i ++ )
            System . out . println ( arr [ i ] );
    }
}

```

ويلاحظ عند تنفيذ هذا البرنامج أن مخرجاته تكون كالتالي :-

```

main
static initializer
constructor
0
1
2

```

وهذا المثال يوضح طريقة تنفيذ برنامج بطريقة بسيطة ، ولناخذ مثالا اخر سر يوضح

كيف تكون مصنفه مبرمجة عبارة عن مصنفه جزئية من مصنفه اخرى (Subclass) حيث أن المصنفه HighNoon هي مصنفه جزئية من Showdown .

```

class ShowDown {
    final static int SHERIFF = 50000 ; // constant class variable
}

public class HighNoon extends ShowDown {
    int good = 20 , bad = 125 . ugly = 53265 ; // a few instance variables
    final static int SHERIFF = 100 ; // constant class variable
    static String welcome = " Have a pleasant stay . " ; // class variable
    static String warning = " Get outta town ! " ; // class variable
    public static void main ( String args [ ] ) {
        System . out . println ( " it's a showdown ... " );
        HighNoon test = new HighNoon ( ) ; // instantiate object
        test . Encounter ( ) ; // invoke Encounter ( ) method
    }
}

```

```

void Encounter ( ) {
System.out.println("Anything greater than"+SHERIFF+"is out ");
    SizeUp ( good );
    SizeUp ( bad );
    SizeUp ( ugly );
}

void SizeUp ( int dude ) {
    System . out . print ( dude + " : " );
    if ( dude > SHERIFF ) { // is it bigger than SHERIFF ?
        System.out.println(warning); // Yes ? Run ' i am out of town
    }
    else {
        System.out.println(welcome); // No ? welcome to numbertville !
    }
}
}

```

ولستطيع أن ندخل بأي عمق في تركيبات المصفقات حيث يمكن أن يصبح المصفق

الجزئي هو مصنف فوق المصنف جزئي آخر منه وهكذا (Nesting hierarchy) ، والمثال

التالي يوضح لنا طريقة لمعرفة بأي مصنف نحن باستخدام (getClass () . getName ()) ، ويلاحظ أن هذه المصنف لها إجراء واحد .



The printClass class

```

class PrintClass {
    int x = 0 ;
    int y = 1 ;
    void printMe ( ) {
        System . out . println ( " X is " + X + " , Y is " + y );
        System . out . println ( " I am an instance of the class " +
            this . getClass ( ) . getName ( ) );
    }
}

```

ولأجل عمل مصنف جزئية من المصنف PrintClass نعمل البرنامج التالي :-

```

class PrintSubClass extends PrintClass {
    int z = 3;
    public static void main ( String args [] ) {
        PrintSubClass obj = new PrintSubClass ();
    }
}

```

Output

وتكون نتيجة هذا البرنامج كالتالي :-

X is 0 , Y is 1

I am an instance of the class PrintSubClass

ويلاحظ أن عملية الطباعة أصبحت داخل المصنفة الجزئية كما تشير المخرجات .

ويمكن أن تستخدم المصنفة الجزئية اجراء بنفس الاسم الذي يستعمله المصنف الجزئي الاصيلي ولي

هذه الحالة فإن الاجراء الموجود داخل المصنفة الجزئية سوف يقوم بإلغاء الاجراء في المصنفة الجزئية (

Method Override ، والمثال التالي يوضح طريقة تجاوز أو إلغاء اجراء من المصنفة الاصلية :-

The PrintSubClass

```

Class PrintSubClass2 extends PrintClass {
    in z = 3;
    void printMe () {
        System.out.println("x is "+ x +", y is" + y +", z is " + z );
        System.out.println("class instance"+ this.getClass(). getName () );
    }
    public static void main ( String args [] ) {
        PrintSubClass2 obj = new PrintSubClass2 ();
        obj . printMe ();
    }
}

```

ويمكن أن نستخدم طريقة للتعاون بين إجرائين في المصنفة الجزئية والاصلية وحتى لو كانوا بنفس الاسم كما في المثال التالي :-

أولا : لنفترض أن الاجراء `printMe` كان في المصنفة الاصلية `PrintClass`

```
void PrintMe () {
    System.out.println("An instance of class"+ this.getClass(
    ).getName ());
    System . out println ( " X is " + " x");
    System . out println ( " Y is " + " y");
}
```

وكان الاجراء `printMe` داخل المصنفة الجزئية `PrintSubClass2` :-

```
void printMe () {
    super . printMe ();
    System . out println ( " Z is " + z);
}
```

وعند تشغيل المصنفة الاصلية فإن المصنفتين سوف تتعاونان لطبع النتائج كالتالي :-

I am an instance of the class `PrintSubClass2`

Output
X is 0
Y is 1
Z is 3

ويمكن أن نسمي عدة اجراءات بنفس الاسم وحتى يمكن أن نغير النوع وعدد مدخلاتها كمدلي المثال التالي :-

The complete MyRect class

```
import java . awt . Point ;
class MyRect {
    int x 1 = 0 ;
    int y 1 = 0 ;
    int x 2 = 0 ;
    int y 2 = 0 ;
```

```

MyRect buildRect ( int x 1 , int y 1 , int x 2 , int y 2 ) {
    this . x 1 = x 1 ;
    this . y 1 = y 1 ;
    this . x 2 = x 2 ;
    this . y 2 = y 2 ;
    return this ;
}

MyRect buildRect ( Point topLeft , Point bottomRight ) {
    x 1 = topLeft . x ;
    y 1 = topLeft . y ;
    x 2 = bottomRight . x ;
    y 2 = bottomRight . y ;
    return this ;
}

MyRect buildRect ( Point topLeft , int w , int h ) {
    x 1 = topLeft . x ;
    y 1 = topLeft . y ;
    x 2 = ( x 1 + w ) ;
    y 2 = ( y 1 + h ) ;
    return this ;
}

void printRect ( ) {
    System . out . print ( " MyRect : < " + x 1 + " , " + y 1 ) ;
    System . out . println ( " , " + x 2 + " , " y 2 + " > " ) ;
}

public static void main ( String args [ ] ) {
    MyRect rect = new MyRect ( ) ;
    System.out.println ("Calling buildRect with coordinates 25 , 25 50 , 50 :");
    rect . buildRect ( 25 , 25 , 50 , 50 ) ;
    rect . printRect ( ) ;
    System . out . println ( " ..... " ) ;
    System.out.println("Calling buildRect w/points(10,10 ),20,20 ):");
    rect . buildRect ( new Point ( 10 , 10 ) , new Point ( 20 , 20 ) ) ;
    rect . printRect ( ) ;
}

```

```

System . out . println ( " ..... " );
System . out . print ( " Calling buildRect w / l point ( 10 , 10 )," );
System . out . println ( " width ( 50 ) and hight ( 50 )" );
rect . buildRect ( new Point ( 10 , 10 ) , 50 , 50 );
rect . printRect ( );
System . out . println ( " ..... " );
}
}

```



ونتائج هذا البرنامج ستكون كالتالي :-

Calling buildRect with coordinates 25 , 25 50 , 50 :

MyRect : < 25 , 25 , 50 , 50 >

.....

Calling buildRect w / points (10 , 10) , (20 , 20) :

MyRect : < 10 , 10 , 20 , 20 >

.....

Calling buildRect w/l point (10,10),width (50) and height (50)

MyRect : < 10 , 10 , 60 , 60 >

.....

4-2- العبارات الرئيسية للغة جافا :-

أن لغة جافا تشبه إلى حد بعيد عبارات لغة سي ولذلك فأنا لن نصرف وقتاً كبيراً في توضيحها ، وسوف نكتفي باستعراض أمثلة عنها . ولنبداً بعباراة (if) الشرطية ولنأخذ حول استخدامها مثالاً يقوم بتوليد رقم عشوائي بين 1 و 100 ومن ثم نقوم بقراءة رقم من المستخدم فإذا كان هذا الرقم يشابه المولد عشوائياً لنجاوب بنعم وبعبكسه يكون الجواب بكلا .

```

import java . awt . * ;
import java . applet . Applet ;
import java . awt . Font ;
public class Applet1 extends Applet {
    TextField guessField = new TextField ( 5 ) ;
    int nextGuess = -1 ;
    int targetNum = ( int )( java . lang . Math . random ( ) * 100 ) - 1 ;
    Fontv f = new Font ( " TimesRoman " , Font . Bold , 24 ) ;

```

```

public void init () {
    setBackground ( Color . white );
    add ( guessField );
}
public void paint ( Graphics g ) {
    String numberStatus = nextGuess + " is correct ";
    g . setFont ( f );
    if ( nextGuess != targetNum ) {
        numberStatus = nextGuess + " is not correct ";
    }
    if ( nextGuess < 1 ) {
        numberStatus = " Guess a number between 1 and 100 ";
    }
    g . drawString ( numberStatus , 20 , 60 );
}

public boolean action ( Event e , Object arg ) {
    if ( e . target instanceof TextField ) {
        try {
            nextGuess = Integer . parseInt ( guessField . getText ( ) );
        }
        catch ( NumberFormatException x ) {
            nextGuess = -1 ;
        }
        repaint ( ) ;
        return true ;
    }
    return false ;
}
}

```

ويمكن تطوير هذا البرنامج لكي نحسن طريقة تعرفنا على الرقم المولد عشوائياً كأن نقول للمستخدم بأنك أقل منه كثيراً أو أعلى منه كثيراً أو أنك الآن حزرتة والمخال التالي يقدم

لنا هذا التحسن وكذلك يوضح استخدام مع عبارة (if) الشرطية استخدام (else) معها ،
والشكل القواعدي للعبارة الشرطية هو :-

```

    if ( condition ) { ..... } else { ..... }

import java . awt . * ;
import java . applet . Applet ;
import java . awt . Font ;
public class Applet 2 extends Applet {
    TextField guessField = new TextField ( 5 ) ;
    int nextGuess = -1 ;
    int targetNum = ( int ) ( java . lang . Math . random ( ) * 100 ) - 1 ;
    Font f = new Font ( " TimesRoman " , Font . Bold , 24 ) ;
public void init ( ) {
    setBackground ( Color . white ) ;
    add ( guessField ) ;
}
public void paint ( Graphics g ) {
    String numberStatus = nextGuess + " is correct " ;
    g . setFont ( f ) ;
    if ( nextGuess != targetNum ) {
        if ( nextGuess < 1 ) {
            numberStatus = " Guess a number between 1 and 100 " ;
        }
        else {
            if ( nextGuess < targetNum ) {
                numberStatus = " nextGuess + " is too high " ;
            }
            else {
                if ( nextGuess < targetNum ) {
                    numberStatus = " nextGuess + " is too low " ;
                }
                else {
                    numberStatus = " nextGuess + " is too high " ;
                }
            }
        }
    }
}
}

```

```

g . drawString ( numberStatus , 20 , 60 ) ;
{

public boolean action ( Event e , Object arg ) {
    if ( e . target instanceof TextField ) {
        try {
            nextGuess = Integer . parseInt ( guessField . getText ( ) ) ;
        }
        catch ( NumberFormatException x ) {
            nextGuess = -1 ;
        }
        repaint ( ) ;
        return true ;
    }
    return false ;
}
}
}

```

والعبارة الأخرى هي العبارة التكرارية (for) التي نستعرضها من خلال المثال التالي حيث نقوم بتحميل مثال حزر الرقم فإذا كان غير صحيحاً نستمر بطبع لحظات لحين التوصل للحل الصحيح. وعبرة
for لنا الشكل القواعدي التالي:--

```

for ( initialization , limiting , increment ) { ..... } ;

```

```

import java . awt . * ;
import java . applet . Applet ;
import java . awt . Font ;

```

```

public class Applet 3 extends Applet {
    TextField userField = new TextField ( 5 ) ;
    int userNum = -1 ;
    Font f = new Font ( " TimesRoman " , Font . Bold , 24 ) ;

```

```

public void init ( ) {
    setBackground ( Color . white ) ;
    add ( userField ) ;

```

```

}

public void paint ( Graphics g ) {
    String outStr = " ";
    int i;
    g . setFont ( f );
    if ( userNum < 1 // userNum > 50 ) {
        outStr = " Enter a number between 1 and 50 ";
    }
    else {
        for ( i = 0 ; i < userNum ; i ++ ) {
            outStr = outStr + " ";
        }
    }
    g . drawString ( outStr , 20 , 50 );
}

public boolean action ( Event e , Object arg ) {
    if ( e . target instanceof TextField ) {
        try {
            userNum = Integer . parseInt ( userField . getText ( ) );
        }
        catch ( NumberFormatException x ) {
            userNum = -1 ;
        }
        repaint ( ) ;
        return true ;
    }
    return false ;
}
}

```

والعبارة التكرارية الاخرى هي عبارة (while) التي لها الشكل القواعدي التالي :-

while (expression) { }

وهناك عبارة دورانية اخرى هي do / while التي لها الشكل القواعدي التالي :-

do { } while (expression) ;

والمثال التالي يوضح استخدام عبارة `do / while` الدورية لنفس مثال حزر الرقم :-

```
import java . awt . * ;
import java . applet . Applet ;
import java . awt . Font ;
public class Applet 5 extends Applet {
    TextField userField = new TextField ( 5 ) ;
    int userNum = -1 ;
    Font f = new Font ( " TimesRoman " , Font . Bold , 24 ) ;
    public void init ( ) {
        setBackground ( Color . white ) ;
        add ( userField ) ;}
    public void paint ( Graphics g ) {
        String outStr = " " ;
        int i = 0 ;
        g . setFont ( f ) ;
        if ( userNum < 1 // userNum > 50 ) {
            outStr = " Enter a number between 1 and 50 " ;}
        else {
            do {
                outStr = outStr + " " ;
                i ++ ;
            } while ( i < userNum ) ;
        }
        g . drawString ( outStr , 20 , 50 ) ;
    }
    public boolean action ( Event e , Object arg ) {
        if ( e . target instanceof TextField ) {
            try {
                userNum = Integer . parseInt ( userField . getText ( ) ) ;
            }
            catch ( NumberFormatException x ) {
                userNum = -1 ;}
            repaint ( ) ;
            return true ;
        }
        return false ;} }
```

ويمكن الخروج من داخل أي عبارة دورانية من خلال استخدام عبارة **break** ، وأخيرا فسان الاختيار المتعدد يكون كم خلال استخدام عبارة **switch** التي لها الشكل القواعدي التالي :-

```
switch ( Number ) {  
    case 1 : ..... ; break ;  
    case 2 : ..... ; break ;  
    :  
    :  
    default : ..... ;  
}
```

ولنأخذ المثال التالي الذي يسمى **Paper , Scissor , Rock** حيث يقوم البرنامج باختيار أحدهما ونحن نحزر أي منهما ، فإذا كان الخزر صحيح يكون الجواب صحيحا وبالعكس .

```
import java . awt . * ;  
import java . applet . Applet ;  
import java . awt . Font ;  
  
public class Applet 7 extends Applet {  
    int appNum = ( int ) ( java . lang . Math . random ( ) . * 3 ) + 1 ;  
    Font f = new Font ( " TimesRoman " , Font . BOLD , 16 ) ;  
    CheckboxGroup userCheckbox ;  
    boolean firstTime = true ;  
    String userChoice ;  
  
    public void init ( ) {  
        setBackground ( Color . white ) ;  
        userCheckbox = new ChickboxGroup ( ) ;  
        add ( new Checbox ( " Rock " , userCheckbox , false ) ) ;  
        add ( new Checbox ( " Scissors " , userCheckbox , false ) ) ;  
        add ( new Checbox ( " Paper " , userCheckbox , false ) ) ;  
    }  
  
    public void paint ( Graphics g ) {  
        int userNum ;  
        string appChoice ;
```

```

g.setFont ( f );
if ( firstTime ) {
g.drawString("Play Rock , Scissors , Paper with me !",20,60);
firstTime = false ;
}
else {
switch ( appNum ) {
case 1 :
appChoice = " Rock " ;
break ;
case 2 :
appChoice = " Scissors " ;
break ;
case 3 :
appChoice = " Paper " ;
break ;
default :
appChoice = " Error " ;
}

switch ( userChoice . charAt ( 0 ) ) {
case ' R ' :
userNum = 1 ;
break ;
case ' S ' :
userNum = 2 ;
break ;
case ' P ' :
userNum = 3 ;
break ;
default :
userNum = 0 ;
}
}

```

```

        if ( appNum == userNum ) {
            g . drawString ( " Tie game - let's play again . " , 20 , 50 ) ;
        }
        else {
            if ((userNum == 1 && appNum == 3 )||( userNum == 2 && appNum == 1 )
                // ( userNum == 3 && appNum == 2 ) ) {
                g.drawString( "I win ! I picked "+ appChoice + " . " , 20 , 60 ) ;
            }
            else {
                g.drawString( " You win ! I picked " + appChoice + " . " , 20 , 60 ) ;
            }
        }
        g.drawString("Reload the page to play another game" . " , 20 , 80 ) ;
    }
}

public boolean action ( Event e , Object arg ) {
    if ( e . target instanceof Checkbox ) {
        userChoice = userCheckbox . getCurrent ( ) . getLabe ( ) ;
        repaint ( ) ;
        return true ;
    }
    return false ;
}
}

```

4-3- تركيبة أدوات التفاعل مع المستخدم :-

أن أهم ما يميز لغة جافا هو وجود العديد من الأدوات والتركيبات المبنية للتفاعل مع المستخدم (GUI) ، وجود مثل هذه الأدوات يجعل بناء صفحات الويب ليس فقط تعطي المعلومة المفهومة بسرعة ، ولكن يمكن اعتبار صفحات الويب على أنها صفحات تفاعلية يمكنها

التحاور مع المستخدم ، ولذلك فإن مستخدمي شبكات الويب اليوم يستطيعون أن يتسببوا ويختاروا وغالباً من خلال صفحات الويب وكذلك يستطيعون ابداء ارائهم والتخاور في اي قضية من خلال هذه الصفحات ، ولذا فإن وجود تركيبات وادوات للتخاور هام جداً وسوف نستعرض في هذه الفقرة اهم هذه الادوات ، وسوف نقوم أولاً باستعراض هذه الادوات أولاً ثم نستعرض كيفية ادارتها .

4-3-1 - كبنسات التفاعل (Buttons) :-

إن لغة جافا تمكننا من خلق كبنسات للتفاعل وعملية اضافتها سهلة جداً تتم باستعمال ايعاز واحد هو (add) وكما يوضحه المثال التالي الذي يقوم بخلق كبنتين :-

```
import java . awt . * ;
import java . applet . * ;
public class MyButtons extends java . applet . Applet {
    public void init ( ) {
        add ( new Button ( " Button 1 " ) ) ;
        add ( new Button ( " Button 2 " ) ) ;
        setStatus ( " Some default layout buttons " ) ;
    }
}
```

أن خلق الكبنسات في المثال اعلاه سوف يضعها في وسط الصفحة ولكن اذا كنا نريد وضعها في مكان معين فهناك طرق عديدة لعمل ذلك ومنها التالي :-

أ - وضع الكبنسات في مواقع الاتجاهات الأربعة :

ويكون ذلك من خلال استخدام العبارة المبنية `setLayout` ، وكما يوضحه المثال التالي الذي يقوم بخلق خمسة كبنسات واحدة لكل اتجاه واحدة في الوسط :-


```

import java . awt . * ;
import java . applet . * ;
public class MyButton2 extends java . applet . Applet {
    public void init ( ) {
        resize ( 200 , 200 ) ;
    }
    public void start ( ) {
        setLayout ( new BorderLayout ( ) ) ;
        add ( " new " , new Button ( " W " ) ) ;
        add ( " East " , new Button ( " E " ) ) ;
        add ( " South " , new Button ( " S " ) ) ;
        add ( " Worth " , new Button ( " N " ) ) ;
        add ( " Center " , new Button ( " C " ) ) ;
        showStatus ( " Button 2 " ) ;
    }
}

```

ب - وضع الكبيات بشكل تسلسلي طولي :-

ويتم التحكم بوضع الكبيات بشكل تسلسلي من خلال استخدام العبارة المبنية
 GridLayout ، والمثال التالي يوضح عملية وضع أربعة كبيات في مساحة محددة بطريقة
 تسلسلية :-

```

import java . awt . * ;
import java . applet . * ;
public class MyButton3 extends java . applet . Applet {
    public void init ( ) {
        resize ( 300 , 200 ) ;
    }
    public void start ( ) {
        setLayout ( new GridLayout ( 15 , 15 ) ) ;
        add ( new Button ( " Button " ) ) ;
        add ( new Button ( " Button " ) ) ;
        add ( new Button ( " Button " ) ) ;
        add ( new Button ( " Button " ) ) ;
        showStatus ( " Grid Buttons " ) ;
    }
}

```

جـ - وضع الكبسات بشكل تسلسلي عرضي :-

ويتم التحكم بوضع الكبسات بشكل مستعرض ، مثلاً من اليسار إلى اليمين من خلال استخدام العبارة المبنية **Flow Layout** ، والمثال التالي يوضح طريقة استخدام هذه العبارة المبنية :-

```
import java . awt . * ;
import java . applet . * ;
public class MyButton4 extends java . applet . Applet {
    public void init ( ) {
        resize ( 100 , 200 ) ;
    }
    public void start ( ) {
        setLayout ( new FlowLayout ( FlowLayout . LEFT ) ) ;
        add ( new Button ( " Button " ) ) ;
        add ( new Button ( " Button " ) ) ;
        add ( new Button ( " Button " ) ) ;
        showStatus ( " FlowLayout Buttons " ) ;
    }
}
```

4-3-2- نوافذ التفاعل :-

إن نافذة التفاعل (**Panel**) هي مساحة يمكن للمستخدم استخدامها بطرق مختلفة مثلاً الرسم الحر فيها أو وضع كبسات في داخلها أو ما شاكل ذلك ، المثال التالي يوضح كيفية وضع كبسات داخل نافذة التفاعل :-

```
import java . awt . * ;
import java . applet . Applet ;
public class MyPanel extends Applet {
    Frame f ;
    public void init ( ) {
        f = new MyWindow ( " Sample Frame " ) ;
        f . resize ( 350 , 150 ) ;
        f . show ( ) ; } }
```

```

public void MyWindow extends Frame {
    MyWindow ( String title ) {
        super ( " Sample Frame " );
        setLayout ( new BorderLayout ( ) );
        Panel topPanel = new Panel ( );
        Panel botPanel = new Panel ( );
        add ( " North " , topPanel );
        add ( " South " , botPanel );
        topPanel . add ( new Button ( " Button 1 " ) );
        botPanel . add ( new Button ( " Button 2 " ) );
    }
}

```

وهناك طريقة أخرى لفتح نوافذ للتفاعل هي باستخدام (canvas) ، التي تقوم بفتح مساحة معينة تشبه طريقة Panel . وطريقة فتح هذه النافذة تتم بشكل عام كالتالي :-

```

class MyCanvas extends canvas {
    void paint ( Graphics g ) {
    }
}

```

4-3-3- قوائم الخيارات :-

يمكن أن نجعل التفاعل مع المستخدم عن طريق قوائم الخيارات وباستخدام الدالة المبنية MenuBar () . والمثال التالي يضع ثلاثة كبسات وعند كبس الأولى تظهر لنا قائمة خيارات :-

```

import java . awt . * ;
import java . applet . Applet ;
public class MyBullDown extends java . applet . Applet {
    Frame f ;
    public void init ( ) {
        f = new MyWindow ( " Sample Frame " ) ;
        f . resize ( 350 , 100 ) ;
    }
}

```

```

class MyWindow extends Frame {
    MyWindow ( String title ) {
        Super ( " Sample Frame " );
        setLayout ( new BorderLayout ( ) );
        Panel myPanel = new Panel ( );
        add ( " Center " , myPanel ( ) );
        myPanel . add ( new Button ( " Yahoo " ) );
        myPanel . add ( new Button ( " InfooSeek " ) );
        myPanel . add ( new Button ( " Webcrawler " ) );
        MenuBar mybar = new Menbar ( );
        Menu m = new Menu ( " Help " );
        m . add ( new MenuItem ( " Item No . 1 " ) );
        m . add ( new MenuItem ( " - " ) );
        m . add ( new MenuItem ( " Item No . 2 " ) );
        m . add ( new MenuItem ( " Item No . 3 " ) );
        mybar . add ( m );
        setMenuBar ( mybar );
    }
}

```

-4-3-4- حقول الكتابة (Text Fields) :-

إن حقول الكتابة هي الاخرى تركيبات التفاعل مع المستفيد وهي على نوعين ، حقول الكتابة (TextField) ومساحات الكتابة (TextArea) وكلاهما يجب أن يكون ضمن

نوافذ التفاعل (Panel) .

```

import java . awt . * ;
import java . applet . Applet ;
public class textarea extends java . applet . Applet {
    public void init ( ) {
        Panel mainPanel = new Panel ( );
        setLayout ( new BorderLayout ( ) );
        add ( " center " , mainPanel );
        mainPanel . add ( new TextArea ( " TextArea " , 5,20 ) );
    }
}

```

وهذا البرنامج سيخلق مساحة يكتب فيها عبارة TextArea ذات أبعاد 20 X 5 سطرًا وعمودًا ، والبرنامج الثاني الذي نقدمه الآن يفتح لنا حقلين للكتابة نكتب في الحقل الاول كلمة TextField 1 وفي الحقل الثاني TextField2 :-

```
import java . awt . * ;
import java . applet . Applet ;
public class tarea 2 extends java . applet . Applet {
    public void init ( ) {
        Panel Panel 1 = new Panel ( ) ;
        Panel Panel 2 = new Panel ( ) ;
        add ( " Center " , Panel 1 ) ;
        add ( " South " , Panel 2 ) ;
        Panel . setLayout ( new GridLayout ( 2 , 1 ) ) ;
        Panel 1 . add ( new TextArea ( " TextArea " , 5 , 50 ) ) ;
        Panel 2 . add ( new TextField ( " TextField 1 " , 15 ) ) ;
        Panel 1 . add ( new TextArea ( " TextField 2 " , 15 ) ) ;
    }
}
```

4-3-5- مربعات الاختيارات (Checkboxes) :-

إن مربعات الاختيار هي أخرى للتفاعل ، والبرنامج التالي يوضح كيفية خلق مربعات اختيار والتي من المفروض أن يختار المستخدم احدهما وعندما يختار احدهما فيكون هو صادق (true) ، وكل الذي لم يتم اختياره هو كاذب (false) :

```
import java . awt . * ;
import java . applet . Applet ;
public class check extends java . applet . Applet {
    Panel myPanel ;
    public void init ( ) {
        myPanel = new Panel ( ) ;
        CheckboxGroup MyChecks = new CheckboxGroup ( ) ;
        myPanel . setLayout ( new GridLayout ( 5 , 1 ) ) ;
        myPanel.add(new checkboxGroup("Archie",MyChecks,false));
        myPanel.add(new checkbox(" Gropher " , MyChecks , false ) ) ;
    }
}
```

```

myPanel . add ( new checkbox ( " WW " , MyChecks , false ) ) ;
myPanel.add( new checkbox ( " Email " , MyChecks , false ) ) ;
myPanel.add( new checkbox ( " WAIS " , MyChecks , false ) ) ;
add ( " Center " , myPanel ) ;
    }
}

```

4-3-6 - قوائم الخيارات النازلة (Choice Lists) :-

يمكن التفاعل مع المستفيد بعرض قوائم خيارات يمكن التأشير عليها حسب رغبته ، ويكون خلق هذه القوائم عن طريق الاجراء المبني (Choice) ، والمثال التالي يوضح هذه الطريقة من التفاعل :-

```

import java . awt . * ;
import java . applet . Applet ;
public class ChoiceTest extends java . applet . Applet {
    Choice ch ;
    public void start ( ) {
        Panel Panel 1 = new Panel ( ) ;
        Panel Panel 2 = new Panel ( ) ;
        setLayout ( new BorderLayout ( ) ) ;
        add ( " North " , Panel 1 ) ;
        add ( " South " , Panel 2 ) ;
        ch = new Choice ( ) ;
        ch . addItem ( " Item 1 " ) ;
        ch . addItem ( " Item 2 " ) ;
        ch . addItem ( " Item 3 " ) ;
        ch . addItem ( " Item 4 " ) ;
        ch . addItem ( " Item 5 " ) ;
        Panel 2 . add ( new Button ( " see Values " ) ) ;
        Panel 1 . add ( ch ) ;
    }
}

```

4-3-7- قوائم ذات مؤلفة مضيفة (Scroll Bars) :-

وهي طريقة اخرى للتفاعل مع المستفيد بحيث تظهر قوائم ذات مؤلفة مضيفة يمكن تحريكها واختيار المدخل المناسب وقد تكون القوائم هي عبارة عن كتابات او حتى صور (Images) ، والبرنامج التالي يقوم بعرض قوائم من النوع ذات الصور ويمكن للمستفيد اختيار احدها .

```
import java . awt . * ;
import java . applet . Applet ;
public class ScrollTest extends java . applet . Applet {
    static Image img ;
    public void init ( ) {
        img = getImage ( getCodeBase ( ) , " myimage . gif " ) ;
        new MyFrame ( ) ; }
class MyFrame extends Frame {
    DrawCanvas cv ;
    Scrollbar hors ;
    Scrollbar vert ;
    public MyFrame ( ) {
        super ( " MyFrame " ) ;
        add ( " Center " , cv = new DrawCanvas ( ) ) ;
        add ( " East " , vert = new Scrollbar ( Scrollbar . VERTICAL , cv . vert _ axis , 0 , 0 , 30 ) ) ;
        add ( " South " , hors = new
        Scrollbar ( Scrollbar . HORIZONTAL , cv . vert _ axis , 0 , 0 , 100 ) ) ;
        resize ( 350 , 110 ) ;
        show ( )
    }
class DrawCanvas extends Canvas {
    int vert _ axis = 0 ;
    int horz _ axis = 0 ;
    public void paint ( Graphics g ) {
        g . translate ( - vert - axis , - hors - axis ) ;
        g . drawImage ( ScrollTest . img , 0 , 0 , this ) ;
    }
}
```

4-3-8- إدارة الفعاليات (Event Handling) :-

إن خلق الكبسات وفتح النوافذ يتطلب معه برمجة الادارة العمليات المختلفة التي ترافق كبس الكبسة أو استخدام النوافذ ، إن ادارة العمليات تقوم بمراقبة العديد من الاحداث والاستجابة لها ، والمثال التالي يقوم بمراقبة ثلاثة كبسات والاستجابة لكل كبسة برقمها :-

```
import java . awt . * ;
import java . applet . * ;
public class test extends java . applet . Applet {
    public void init ( ) {
        resize ( 100 , 300 ) ; }
    public void start ( ) {
        setLayout ( new FlowLayout ( FlowLayout . LEFT ) ) ;
        add ( new Button ( " Button 1 " ) ) ;
        add ( new Button ( " Button 2 " ) ) ;
        add ( new Button ( " Button 3 " ) ) ; }
    public boolean handleEvent ( Event evt ) {
        switch ( evt . id ) {
            case ( Event . ACTION_EVENT ) : {
                if ( evt . arg == " Button 1 " ) ;
                    System . out . println ( " Button 1 " ) ;
                    return true ;
                } else
                if ( evt . arg == " Button 2 " ) {
                    System . out . println ( " Button 2 " ) ;
                    return true ;
                } else
                if ( evt . arg == " Button 3 " ) {
                    System . out . println ( " Button 3 " ) ;
                    return true ;
                } else
                    return false
            }
        }
        default : return false ;
    }
}
```


ويلاحظ أن ادارة العمليات تمت داخل اجراء هو (`handleEvent`) ويمكن ادارة العمليات أيضا باستخدام اجراء آخر هو (`action`) ، والمثال التالي يوضح استخدام هذا الاجراء :-

```
import java . awt . * ;
import java . applet . Applet ;
public class tarea 3 extends java . applet . Applet {
    TextField fld 1 ;
    TextField fld 2 ;
    TextField ta 1 ;
    public void init () {
        Panel Panel 1 = new Panel () ;
        Panel Panel 2 = new Panel () ;
        Panel Panel 3 = new Panel () ;
        add ( " North " , Panel 1 ) ;
        add ( " Center " , Panel 2 ) ;
        add ( " South " , Panel 3 ) ;
        Panel 1 . add ( new Button ( " see values " ) ) ;
        Panel 2 . add ( ta 2 = new TextArea ( " TextArea " , 5 , 50 ) ) ;
        Panel 3 . add ( fld 1 = new TextField ( " TextField 1 " , 15 ) ) ;
        Panel 3 . add ( fld 2 = new TextField ( " TextField 2 " , 15 ) ) ;
    }
    public boolean action ( Event evt , Object arg ) {
        String label = ( String ) arg ;
        if ( label == " See values " ) {
            System . out . println ( " TextArea : " + ta 1 . getText () ) ;
            System . out . println ( " " ) ;
            System . out . println ( " TextField 1 : " + fld 1 . getText () ) ;
            System . out . println ( " " ) ;
            System . out . println ( " TextField 2 : " + fld 2 . getText () ) ;
            return true ;
        } else
            return false ;
    }
}
```

ويمكن كتابة اجراءات لادارة بعض العمليات الخاصة مثل ادارة جهاز الفأرة (mouse) والبرنامج التالي يوضح بعض الاجراءات التي تتفحص حركة جهاز الفأرة او الضغط على أحد أزراره :-

```
import java . awt . * ;
import java . applet . Applet ;
public class MouseTest extends Applet {
    public void init ( ) {
        System . out . println ( " Initializing Applet .... " ) ; }
    public boolean mouseDown ( Event event , int x , int y ) {
        System . out . println ( " Mouse button click . " ) ;
        return true ; }
    public boolean mouseEnter ( Event event , int x , int y ) {
        System . out . println ( " Mouse Enteres applet area " ) ;
        return true ; }
    public boolean mouseExit ( Event event , int x , int y ) {
        System . out . println ( " Mouse exited applet area " ) ;
        return true ; }
    public boolean mouseDrag ( Event event , int x , int y ) {
        System . out . println ( " Mouse being dragged " ) ;
        return true ; }
    public boolean mouseMove ( Event event , int x , int y ) {
        System . out . println ( " Mouse moving " ) ;
        return true ; }
}
public class MouseXY extends Applet {
    public boolean mouseDown ( Event event , int x , int y ) {
        System.out.println("Mouse click at coordinate : "+x+","+y");
        return true ;
    }
}
```

ونلاحظ أن المصنف Mouse XY هي مكررة ولكنها فقط لطبع الموقع الذي فيه ، ثم ضغط زر جهاز الفأرة .

الفصل الخامس

دارة الفعاليات المتزامنة

الفصل الخامس

إدارة العمليات المتزامنة

5-1- مفهوم خيوط التعاون التنفيذية المتزامنة :-

إن لغة جافا تمكن من برمجة العمليات بشكل متزامن (multitasking) من خلال تعريف كل عملية من خلال خيط مستقل للتنفيذ (thread) ويمكن تنفيذ جميع العمليات بشكل متزامن من خلال التعاون بين هذه الخيوط التنفيذية وإعطاء كل خيط زمن تنفيذ (time slice) خاص به ، وتفيد العمليات المتزامنة في العديد من التطبيقات وأساليب البرمجة حيث يمكن إجراء عمليات في خلفية البرامج التي ننفذها فعلى سبيل المثال يمكننا استمساخ برامج كبيرة ونحن ننفذ برنامجا معينا وفي الواقع أن عمليات التنفيذ المتزامن تكبد في العديد من التطبيقات المعروفة مثل :-

- استخدامها في خلق صور متحركة (animation)
 - استخدامها في بث ومعالجة الأصوات (Voice Manipulation)
 - استرجاع وتحديث البيانات في خلفية البرامج & (updating Retrieving)
 - انتظار استجابة داخل شبكة الانترنت في خلفية البرنامج (Waiting)
- وتمكننا لغة جافا من تعريف خيوط التنفيذ المتوازي بطريقتين مختلفتين :-
- أ - المصنفة البرمجية التي تعمل بتزامن مع عمليات قياسية (Runnable) :-
- وتعريف المصنفة البرمجية بهذه الطريقة يتطلب إضافة العبارة (implements Runnable) وكالتالي :-

```
class MyApplet extends java . applet . Applet implements Runnable
{
    .....
    public void run ( ) {
        // body of a thread
    }
}
```

ويلاحظ بأننا في هذه الحالة نحتاج إلى وجود إجراء باسم (run) الذي نطبع فيه خوارزمية الخيط المترامن ، وهذه الطريقة تمكننا من خلق خيط تنفيذي يشتغل بشكل مترامن مع بقية عمليات صفحات الويب (Browser) ، ونحتاج لخلق خيط تنفيذي مترامن في هذه الطريقة إلى تعريف متغير (object) على أنه من النوع (Thread) ، مع إجرأين للبداية (start) الذي يقوم بمناداة إجراء (run) وإجراء للنهاية (stop) كالآتي :-

```
Thread threadObj ;
...
public void start ( ) {
    if ( threadObj == null ) {
        threadObj = new Thread ( this , " My Thread " ) ;
        threadObj . start ( ) ;
    }
    ...
    public void stop ( ) {
        threadObj . stop ( ) ;
        threadObj = null ;
    }
}
```

وكمثال لاستخدام هذه التقنية ندرج مثالا لطباعة الوقت على صفحة الويب بشكل مترامن مع بقية العمليات التي يمكن أن نجريها مع معالجة صفحات الويب :

```
import java . awt . Graphics ;
import java . util . Date ;
public class Clock extends java . applet . Applet implements
Runnable {
    Thread ClockThread ;
    public start ( ) {
        if ( ClockThread == null ) {
            ClockThread = new Thread ( this , " Clock " ) ;
            ClockThread . start ( ) ;
        }
    }
}
```

```

public void run ( ) {
    while ( ClockThread . = null ) {
        repaint ( ) ;
        try {
            Clock Thread . sleep ( 1000 ) ; }
        catch ( InterruptedException e ) {}
    }
}

public void paint ( Graphics ) {
    date now = new Date ( ) ;
    g drawing(now.getHours()+"now.getMinutes()+":"now.getSeconds(),5, 10 ) ;
}

    public void stop ( ) {
        clockThread . stop ( ) {
            clockThread = null ;
        }
    }
}

```

ب - المصنفة البرمجية التي هي مصنفة جزئية من الخيط التنفيذي (Thread)
وهذه الطريقة هي أشمل من الطريقة السابقة حيث أنها تمكنا من تعريف أكثر من خيط برمجي واحد يمكن أن يعمل فقط مع معالج صفحات الويب ، وهذه الطريقة تقوم بتعريف المصنفة البرمجية التي تحتوي نص الخيط البرمجي على أنها مصنفة جزئية من المصنفة العامة (java . lang . Thread) :-

```

class MyThread extends java . lang . Thread {
    .....
    public void run ( ) {
        .....
    }
}

```

حيث يمكن خلق أكثر من خيط تنفيذي بالشكل التالي :-

```
import java . applet . Applet ;
class CreatTwoThreads extends Applet {
    public void CreatTwoThreads ( ) {
        new CreatThreads 1 ( ) . start ( ) ;
        new CreatThreads 2 ( ) . start ( ) ;
    }
}
```

```
class CreatThread 1 extends Thread {
    .....
    public run ( ) {
        .....
    }
}
class CreatThread 2 extends Thread {
    public run ( ) { .....}
}
}
```

وكمثال على خلق أكثر من خيط تنفيذي ندرج المثال التالي الذي يقوم بخلق ثلاثة خيوط تنفيذية تعمل سوياً وكل خيط تنفيذي فيه عبارة دورانية تدور خمس مرات تقوم بطبع أسم الخيط التنفيذي .

```
class EZThread extends Thread {
    public EZThread ( String str ) {
        super ( str ) ; // pass up to Thrad constructor
    }
    public void run ( ) {
        for ( int i = 0 ; i < 5 ; i ++ ) {
            System . out . println ( i + " " + getName ( ) ) ;
            try {
                sleep ( ( int ) ( Math . random ( ) * 500 ) ) ;
            } catch ( InterruptedException e ) {}
        }
        System . out . println ( getname ( ) + " Has Expired " ) ;
    }
}
```



```

class EZTest {
    public static void main ( string arg [ ] ) {
        new EZThread ( " Hickory " ). start ( ) ;
        new EZThread ( " Dickory " ). start ( ) ;
        new EZThread ( " Dock " ). start ( ) ;
    }
}

```

وعند تشغيل المصنفة EZTest فإن مخرجات هذا البرنامج ستكون كالتالي :-

> java EZTest
 علما بأن الخيوط التنفيذية تعمل وفق الوقت الذي يتفرغ فيه المعالج وليس هناك أي ترتيب لتنفيذ أي خيط قبل الآخر .

```

Hickory
Dickory
Dock
Dock
Dickory
Dickory
Dock
Dickory
Hickory
Dickory
Dock
Dickory has Expired
Hickory
Hickory
Dock
Dock has Expired
Hickory has Expired

```

5-2- إجراءات أخرى للتحكم بالخيوط التنفيذية :-

إن الخيوط التنفيذية يمكن التحكم بأساليب تنفيذها مثل إعطاء بعضها أولوية أكبر من الآخر عن طريق إجراء مبني هو (`setPriority`) الذي عند استخدامه يمكن إعطاء الخيط التنفيذي أعلى أو أقل أولوية للتنفيذ :-

```
Thread myThread ;
public void init ( ) {
    myThread = new Thread ( this ) ;
    myThread . setPriority ( Thread . MAX _ PRIORITY )
;
    ...
}
```

وهناك العديد من الإجراءات المبنية الأخرى التي هي متوفرة لدى المصنفة التي تساعد على التنفيذ المتزامن (Thread) وكما يوضحه الجدول (1-5) .

Class java . lang Thread

Constructor	Signature	Description
Thread	public Thread ()	Constructs a new thread. Threads created this way must override their <code>run ()</code> method to do anything . An example illustrating this method is shown in the sidebar " Using the Thread () Constructor " .
Thread	Public Thread(Thread Group group, Runnable target)	Constructs a new thread which applies the <code>run ()</code> method of the specified target. Parameter : target-object whose <code>run ()</code> method is called
Thread	public thread (ThreadGroup group, Runnable target)	Constructs a new thread in the specified thread group that applies the <code>run ()</code> method of the specified target. Parameter : group - the thread group target-object whose <code>run ()</code> method is called.
Thread	public thread (String name)	Constructs a new thread with the specified name . Parameter : name - name of the new thread.
Thread	public thread (ThreadGroup group, String name)	Constructs a new thread in the specified thread group with the specified name. Parameter : group - thread group name - name of the thread
Thread	public Thread (Runnable target, String name)	Constructs a new thread in the specified name and applies the <code>run ()</code> method of it specified target. Parameter : target - object whose <code>run ()</code> method is called name - name of the new thread.
Thread	public Thread (ThreadGroup group, Runnable target, String name)	Constructs a new thread in the specified thread group with the specified name and applies the <code>run ()</code> method of the specified target.

currentThread	<code>public static Thread currentThread ()</code>	Parameter : group - thread group target - object whose - thread run () method is call name - name of the thread . Returns a reference to the currently executing thread object.
yield	<code>public static void yield</code>	Causes the currently executing Thread object to yield. If there are other runnable thread they will be scheduled next.
sleep	<code>public static void sleep (long millis) throws InterruptedException</code>	Causes the currently executing thread to sleep for the specified number of milliseconds. Parameter : millis - length of time to sleep in milliseconds.
sleep	<code>public static void sleep (long millis, int nanos throws InterruptedException</code>	throws InterruptedException if another thread has interrupted this thread. Sleep for the specified number to nanoseconds. Parameter : millis - length of time to sleep in milliseconds . nanos-0-99999 additional nanoseconds to sleep.
start	<code>public synchronized void start ()</code>	Throws InterruptedException if another thread has interrupted this thread. Start this thread. This will cause the run () method to be called. This method will return immediately . Throws Illegal thread state Exception if the thread was already started.
run	<code>public void run ()</code>	The actual body of this thread. This method is called after the thread is started . you must either override this method by subclassing class Thread , or you must create the thread with a runnable target .
stop	<code>public final void stop ()</code>	Stops a thread by tossing an object. By default , this routine tosses a new instance of Thread Death to the target thread. Thread Death is not actually a subclass of Exception, but is a subclass of Object. Users should not normally try to catch it unless they must do some extraordinary cleanup operation. If thread Death is caught, it's important to rethrow the object so that the thread will actually die. The top-level error handler will not print out message if it falls through.
stop	<code>public final synchronized void stop (Throwable e)</code>	Stops a thread by tossing an object. Normally, users should just call the stop () method without any argument. However , in some exceptional circumstances used by the stop () method to kill a thread, another object is tossed. Thread Death is not actually a subclass of Exception, but is a subclass of Throwable.
interrupt	<code>public void interrupt ()</code>	Parameter : e - object to be throws . Sends an interrupt to a thread.
isInterrupted	<code>public static boolean isInterrupted ()</code>	Asks if you have been interrupted.
isInterrupted	<code>public boolean isInterrupted ()</code>	
destroy	<code>public void destroy ()</code>	Destroys a thread without any lean - up : in other words, just tosses its state; any monitors it has locked remains locked . A last resort.
suspend	<code>public final void suspend ()</code>	Suspends this thread's execution.
resume	<code>public final void resume ()</code>	Resumes this thread's execution. This method is only valid after suspend () has been invoked.
setPriority	<code>public final void setPriority (int new Priority)</code>	Sets the thread's priority. Throws Illegal Argument Exception if the priority is not within the range MIN-PRIORITY,MAX-PRIORITY.
getPriority	<code>public final int getPriority ()</code>	gets and returns the thread's priority .

setName	<code>public final void setName (String name)</code>	Sets the thread's name. Parameter: name-new of the thread's.
getName	<code>public final String getName ()</code>	Gets and returns this thread's name.
getThreadGroup	<code>public final ThreadGroup getThreadGroup ()</code>	Gets and returns this thread group.
activeCount	<code>public static int activeCount ()</code>	Returns the current number of active threads in this thread group.
enumerate	<code>public static int enumerate (Thread tarray [])</code>	Copies, into the specified array, references to every active thread in this thread's group. Returns the number of threads put into the array.
countStackFrames	<code>public int countStackFrames ()</code>	Returns the number of the stack frames in this thread. The thread must be suspended when this method is called.
join	<code>public final synchronized void join (long millis) throws InterruptedException</code>	Throws <code>IllegalThreadStateException</code> if the thread is not suspended. Waits for this thread to die. A time-out in milliseconds can be specified. A time-out of 0 (zero) milliseconds means to wait forever. Parameter: millis - time to wait in milliseconds Throws <code>InterruptedException</code> if another thread has interrupted this one Waits for the thread to die, with more precise time.
Join	<code>public final synchronized void join (long millis, int nanos) throws InterruptedException</code>	Throws <code>InterruptedException</code> if another thread has interrupted this thread.
Join	<code>public final void join () throws InterruptedException</code>	Waits forever for this thread to die.
dumpStack	<code>public static void dumpStack ()</code>	Throws <code>InterruptedException</code> if another thread has interrupted this thread. A debugging procedure to print a stack trace for the current thread.
setDaemon	<code>public final void setDaemon (boolean on)</code>	Marks this thread as a daemon thread or a user thread. When there are only daemon threads left running in the system, Java exits. Parameter: on - determines whether the thread will be a daemon thread
isDaemon	<code>public final boolean isDaemon ()</code>	Throws <code>IllegalThreadStateException</code> if the thread is active. Returns the daemon flag of the thread.
checkAccess	<code>public void checkAccess ()</code>	Check whether the current thread is allowed to modify this thread. Throws <code>SecurityException</code> if the current thread is not allowed to access this thread group.
toString	<code>public String toString ()</code>	Returns a string representation of the thread, including the thread's name, priority, and thread group. Overrides <code>toString</code> in class <code>Object</code> .
MIN-PRIORITY	<code>public final static int MIN-PRIORITY</code>	The minimum priority that a thread can have. The most minimal priority is equal to 1.
NORM-PRIORITY	<code>public final static int NORM-PRIORITY</code>	The minimum priority that is assigned to a thread. The default priority is equal to 5.
MAX-PRIORITY	<code>public final static int MAX-PRIORITY</code>	The maximum priority that a thread can have. The maximal priority value a thread can have is 10.

جدول 1-5 : الإجراءات المبينة على الخوارزمية المتزامنة .

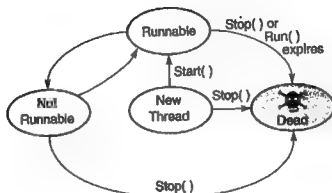
ان فهم أي خيط تنفيذي يحتاج الى استيعاب الفقرات التالية :-

1. مكان خوارزمية الخيط التنفيذي (Thread Body) :-

والذي يكتب داخل إجراء `run()`.

2. حالات تنفيذ الخيط التنفيذي (Thread Body) :-

خلال حياة الخيط التنفيذي يمر التنفيذ خلالها بتحويلات يلخصها الشكل (1-5) :



شكل (1-5) : حالات تنفيذ الخيط التنفيذي .

والحالة الاولى التي يبدأ فيها ي خيط تنفيذي بالحياة هي مرحلة الخلق (New Thread) التي تبدأ فعلا عندما ننفذ العبارة التالية :

`Thread myThread = new Thread (this) ;`

وبالتأكيد فنحن نحتاج الى اعطاء الخيط التنفيذي خوارزمية لكي يقوم بتنفيذها

وبدون هذه الخوارزمية فنحن نستطيع بدايته (start) وتوقيفه (stop) فقط . اما الحالة الثانية التي يمكن الانتقال اليها فهي حالة التشغيل (Runnable) وهي الحالة بعد تنفيذ الابعاز (start) . والحالة الثالثة يمكن

للخيط التنفيذي الدخول فيها هي حالة عدم الاستغال (Not Runnable) التي يدخلها الخيط لاسباب التالية :

- تم توقيفه مؤقتا (suspended) .

- تم اخلاله في مرحلة انتظار زمني (sleep) .

- تم اخلاله في مرحلة انتظار تحقق شرط (waiting) .

- تم توقيفه من قبل خيط تنفيذي آخر (blocked) .

وانتقال لحالة التوقيف بسبب تنفيذ الاجراءات التالية (suspend) او (sleep (join () ، اما الحالة الاخيرة وهي حالة موت الخيط تتم بعد الانتهاء من تنفيذ اجراء التشغيل (run (التابع له وتنفيذ اجراء (stop (ايقافه تماما :

3. اولوية التنفيذ (Thread Priority) .

ويمكن اعطاء كل خيط تنفيذي اسبقية بواسطة الاجراء SetPriority ولكن هنا سيضمن اعتمادنا كليا على تقدير المخطط الداخلي والذي يقوم باعطاء كل خيط تنفيذي وقت معين (Java Runtime Scheduler) (Time Slice) ولضمان اعطاء اجراءنا وقتا متساويا مع بقية الخيوط البرمجية يمكن استخدام اجراء (yield .

4. الخيط الرئيسي (Daemon Thread) .

وهو الخيط الذي يسمى احيانا (task master) الذي يستطيع تنفيذ خيوط اخرى بواسطة وبصورة اعتيادية يكون هذا الخيط قابل للتنفيذ بصورة دائمة لكونه يحتوي على اجراء (run (في دلالة عبارة تكرارية غير منتهية (Infinite loop) ويمكننا تعريف اي خيط على انه رئيسي اذا استخدمنا العبارة المبنية التالية :

myThread . setDaemon (true) ;

وكذلك يمكن فحص اي خيط على انه رئيسي ام لا من خلال تنفيذ العبارة :

mythread . isDaemon () ;

5. المجاميع الخيطية (Thread Groups) :

أن المجاميع الخيطية هي مصنفات مبرمجة تكون فيها المصنفة (Thread (عبارة عن مصنفة جزئية وكذلك فإن استخدام المصنفة Thread Group تمكننا من التعامل مع أكثر من خيط برمجي وبرمجة العمليات بين الخيوط البرمجية المتعددة .

جدول 2-5 : يوضح الاجراءات الاساسية داخل المصنفة للمجاميع الخيطية .

Class java . lang . ThreadGroup		
Constructor	Signature	Description
Thread Group	public Thread Group(String name)	creates a new thread group. Its parent will be the thread group of the current thread. Parameter: name-name of the new thread group created.
Thread Group	public Thread Group (thread Group parent, String name)	Creates a new thread group with a specified name in the specified thread group. Parameter: parent-specified parent thread group name-name of the new thread group being created. Throws NullPointerException if the given thread group is equal to null .
Method		
activeCount	public synchronized int activeCount ()	Returns an estimate of the number of active threads in the thread group.
activeGroupCount	public synchronized int active GroupCount ()	Returns an estimate of the number of active groups in the thread group.
checkAccess	public final void checkAccess ()	Checks to see if the current thread is allowed to modify this group. Throws Security Exception if the current thread is not allowed to access this thread group.
destroy	public final synchronized void destroy ()	Destroy a thread group. This does NOT stop the threads in the thread group. Throws Illegal thread State Exception if the thread group is not empty, or if it was already destroyed.
enumerate	public int enumerate (Thread list [])	Copies, into the specified array, references to every active thread in this thread group. You can use the active Count () method to get an estimate of how big the array should be. Parameter: list - an array of threads Returns the number of threads put into the array.
enumerate	public int enumerate (Thread list [])	Copies, into the specified array, references to every active thread in this thread group. You can use the active Count () method to get an estimate of how big the array should be. Parameter: list - an array of threads groups Returns the number of thread groups placed into the array.
enumerate	public int enumerate (Thread list [])boolean recurse	Copies, into the specified array, references to every active thread in this thread group. You can use the active Count () method to get an estimate of how big the array should be. Parameter: list - an array of threads recurse-a boolean indicating whether a thread has reappeared. Returns the number of threads placed into the array.
getMaxPriority	public final int get Max Priority ()	Gets the maximum priority of the group. Thread that are part of this group cannot have a higher priority than the maximum priority.
get Name	public final String getName ()	Gets the name of this thread group.
get Parent	public final Thread Group get Parent ()	Gets the parent of this thread group
is Daemon	public final boolean is Daemon ()	Returns the daemon flag of the thread group. A daemon thread group is automatically destroyed when it is found empty after a thread group of thread is removed from it .

list	public synchronized void list ()	Lists this thread group. Useful for debugging only.
parent Of	public final boolean 'parent Of (Thread Group g)'	Checks to see if this thread group is a parent of, or is equal to, another thread group. Parameter : g - thread group to be checked Returns true if this thread group is equal to, or is the parent of, another thread group, false otherwise.
resume	public final synchronized void resume ()	Resume all the threads in this thread group and all of its sub-groups.
set Daemon	public final void set Daemon (boolean daemon)	Changes the daemon status of this group. Parameter : daemon - daemon boolean which is to be set .
set Max Priority	public final synchronized void set Max Priority (int pri)	Sets the maximum priority of the group. Threads that are already in the group can have a higher priority than the set maximum. Parameter : pri - priority of the thread group .
stop	public final synchronized void stop ()	Stops all the threads in this thread group and all of its sub - groups.
suspend	public final unsynchronized void suspend ()	Suspends all the threads in this thread group and all of its sub - groups.
to String	public String to String ()	Returns a string representation of the thread group . Overrides to String in class Object.
uncaught Exception	public void uncaught Exception (Thread t Throwable e)	Called when a thread in this group exists because of an uncaught exception. Parameters : t - the thread e - a Throwable object.

جدول : 5-2 : الاجراءات المبينة داخل مجاميع الخيوط

5-3 - التنفيذ المتعدد لخيوط التنفيذ المتزامنة (Multithreaded) :-

إن برمجة خيوط تنفيذ متعددة يعرف بالتنفيذ المتعدد (Multithreaded Programming) ، وهذا يتطلب ترتيب عمليات التعاون بين الخيوط التنفيذية المتعددة (Synchronization) ، بحيث يمكن تنظيم التعاون بين أي لجزائين أو عمليتين أو خيوطين بشكل منظم ، فمثلا عندما يتعاون خيطان تنفيذيان لاستهلاك مخزن احدهما (منتج) والثاني (مستهلك) ينتهي فإن المستهلك ينتظر وعندما يمتلئ المخزن يتوقف المنتج عن الانتاج وينتظر لحين انتهاء المستهلك من الانتاج وهكذا ويمكن تأمين هذا التعاون بأن نضع الكلمة المفتاحية (synchronized) قبل اسم الاجراء او الخيط وكما يلي :

```
synchronized void myMethod () {  
}
```

ويمكن أن نطلق على الاجراء الذي يحوي عبارة synchronized بأنه اجراء مراقب (monitor) لانه يستطيع أن ينادي في داخله اجراء باسم (bit) ، الذي يقوم بإيقاف كافة الخيوط الأخرى (blocked) لحين الانتهاء من التنفيذ ، وهناك ملاحظة عامة حيث يمكن وضع عبارة (synchronized) لجعل أي شيء أو متغير من النوع الذي يقوم بتنفيذ خيط واحد في الوقت الواحد ، وهنا يجب الانتباه إن عملية تنظيم التنفيذ واستخدام عبارة synchronized قد يخلق

حالات من الانتظار غير منتهية ، وهي ما يعرف بحالة الانتظار القاتل (Dead lock) التي يجب تجنبها وبالذات بعدم استخدام عبارة (lock) لمدة طويلة وكذلك عدم منادات اجراء مترامن (synchronized Method) داخل اجراء مترامن آخر ، ويمكن تنظيم عملية التزامن من خلال استخدام اجراءات wait و notify ، والمثال التالي يمثل تزامن ثلاثة خيوط برمجية والتعاون بينهما باستخدام الاجراءات اعلاه :

```
import java . awt . * ;
public class DoesNotify App extends java . applet . Applet implements
Runnable {
    Thread      thread 1 ;
    Thread      thread 2 ;
    Thread      thread 3 ;
    int myArray [ ] = new int [ 10 ] ;
    public void init ( ) {
        thread 1 = new Thread ( this , " thread 1 " ) ;
        thread 2 = new Thread ( this , " thread 2 " ) ;
        thread 3 = new Thread ( this , " thread 3 " ) ;
    }
    public void start ( ) {
        thread 1 . start ( ) ;
        thread 2 . start ( ) ;
        thread 3 . start ( ) ;
    }
    public void run ( ) {
        if ( Thread . current Thread ( ) == thread 1 ) {
            while ( true ) {
                synchronized ( myArray ) {
                    for ( int I = 0 ; I < 10 ; I ++ ) I
                    myArray [ I ] = 0 ;
                    system . out . println ( " Update 1 " + I ) ;
                }
            }
        }
        doNotify ( ) ;
        try {
            thread . sleep ( 1900 ) ;
        }
    }
}
```

```

    } catch { InterruptedException ignored } {
        system.out.println ( " Can't Sleep " );
        return ;
    }
}

    if ( thread . current thread ( ) = thread 2 ) {
        while ( true ) {
            synchronized ( myArray ) {
                for ( int I = 0 , I < 10 ; I ++ ) {
                    myarray [ I ] = 1 ;
                    System . out . println ( " update 2 " + I ) ;
                }
            }
            DoNotify ( ) ;
            try {
                Thread . sleep ( 2100 ) ;
            } catch { InterruptedException inored } {
                System . out . println ( " Can't sleep " )
            }
            return ;
        }
    }

    if ( thread . current Thread ( ) == thread 3 ) {
        while ( true ) {
            DoWait ( ) ;
            for ( int i = 0 ; i < 10 ; i ++ )
                system.out.println( "Value" + i + " + myArray [ i ] );
        }
    }

    Synchronized private void DoNottify ( ) {
        notify ( ) ;

        Synchronized private void DoWait ( ) {

```

```

        system . out println ( " Well I'am Waiting " ) ;
    try {
        wait ( ) ;
    } catch { Exception Et } {
        System . out . println ( " Thread has been
        Interrupted " ) ;
    } }

```

ولنأخذ مثالا تطبيقياً آخر على عمليات التزامن ، حيث يوضح هذا البرنامج تصميم لعبة باسم (PingPong) وخوارزمية هذه اللعبة هو خلق خيطين يتعاونان كالتالي :-

```

If it is my turn ,
    note whose turn it is next .
    then write PING ,
    and then notify anyone waiting,
otherwise
    wait to be notified .

```

أن البرنامج الذي يحقق هذه الخوارزمية هو كالتالي:

```

// The " Player " class
public class PingPong {
// state variable identifying whose turn it is .
    private String whoseTurn = null ;
    public synchronized boolean hit ( String opponent ) {
        String x = thread . current Thread ( ) . getName ( ) ;
        if ( whoseTurn == null ) {
            whoseTurn = x ;
            return true ;
        }
        if ( whoseTurn . compareTo ( " DONE " ) == 0 )
            return false ;
        if ( whoseTurn . compareTo ( " DONE " ) == 0 )
            whoseTurn = opponent ;
            notifyAll ( ) ;
            return false ;
        }
        if ( x . compareTo ( whoseTurn ) == 0 {

```

```

        System.out.println ( " PING ! (" + x + " )" );
        whoseTurn = opponent ;
        notifyAll () ;
    } else {
    try {
        long t1 = system.currentTimeMillis () ;
        wait ( 2500 ) ;
        if (( system.currentTimeMillis () . t 1 ) > 2500 ) {
            System.out.println ( " ***** TIMEOUT ! " + x +
                " is waiting for + whoseTurn + + whose + " tp play " );
        }
    } catch ( InterruptedException e ) { }
    }
    return true ; // keep playing }

// The " Player " class
public class Player implements Runnable {
    PingPong myTable ;    // Table where they play
    String myOpponent ;
    public Player ( String opponent , PingPong table ) {
        myTable = table
        myOpponent = opponent ;
    }
    public void run () {
        while ( myTable . hit ( my Opponent ) );
    }
}

public class Game {
public static void main ( String args [ ] ) {
    PingPong table = new Thread ( new Player ( " bob " , table ) );
    thread alice = new Thread ( new Player ( " alice " , table ) );
        alice.SetName ( " alice " ) ;
        bob . setName ( " bob " ) ;
        alice .start () ;    // alice starte playing
        bob . start () ;    // bob starts playing
    try {

```

```

// Wait 5 seconds
thread . currentThread ( ) . sleep ( 5000 ) ;
} catch ( InterruptedException e ) { }
table . hit ( " DONE " ) ; // cause the players to quit their threads .
try {
    thread . currentThread ( ) . sleep ( 100 ) ;
} } }

```

علماً بأن نتائج تشغيل هذه اللعبة هو كالتالي :

```

PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )
PING !      ( alice )
PING !      ( bob )

```


الفصل السادس
هياكل البيانات
باستخدام جافا

الفصل السادس

هياكل البيانات واستخدام جافا

6-1-1 أساسيات هياكل البيانات في جافا :-

يمكننا تعريف هياكل البيانات باستخدام لغة جافا بطريقة مختلفة عن اللغات السابقة ، حيث لا يوجد في جافا هذه المؤشرات (pointers) ويمكننا تعريفها كالتالي :-

إن هياكل البيانات هي مجموعة من العقد (nodes) التابعين لنفس المصفوفة المبرمجة (class) مرتبة بطريقة معينة وعملية الوصول إليها معرفة مسبقاً ، والعقدة هي عبارة عن شيء مستقر في الذاكرة (object) وقد يكون عبارة عن متغير بسيط أو مركب ، ولغة جافا تقوم بتعريف بعض هياكل البيانات بطريقة مبنية (built - in) مثل هيكل المصفوفة (array) وقسم منها مزود من قبل مصنفات مبرمجة خاصة مثل المصفوفة المتقدمة (java.util.Vector) (او المكمنس (java.util.Stack) او المصفوفة المقتننة)

(java.util.Hashtable) او المجاميع (java.util.BitSet) او مصفوفة القاموس (java.util.Dictionary) او الملفات المتسلسلة (java.io.File-Stream) او الملفات ذات الوصول الاحتياطي (java.io.RandomAccessFile) ، أما القسم الغير مبني او المزود بمصنفات مبرمجة فيتم بناءه من قبل المبرمج ، وهناك خواص خاصة ترتبط بكل انواع هياكل البيانات أهمها :-

1. علاقة العقدة بالنمسة للعقد الأخرى (Relationship to nodes) .
2. كيفية بناء العقدة الرئيسية (header node) .
3. كيفية اضافة عقدة معينة الى الهيكل وفي اي موقع (deleting nodes) .
4. اتجاه استرجاع البيانات من الهيكل (.. forward, backwards, etc) .

وبصورة عامة تمثيل اي هيكل بيانات يتم بطريقتين اثنتين ، الاولى باستخدام المصفوفة (array) والثانية باستخدام عقد مربوطة ببعض عن طريق مرجعية معينة (references) ويمكن تقسيم انواع هياكل البيانات الى نوعين رئيسيين ، احدهما خطي (linear) والثاني لا خطي (Nonlinear) .

2-6- هياكل البيانات الخطية (Linear Data Structures) :-

إن هياكل البيانات الخطية هي تلك الهياكل التي ترتب عقدها بشكل متمسك ولذلك فلها عنصر أولي وعنصر آخر وعنصر سابق وعنصر لاحق ، وبطريقة بسيطة يمكن تمثيل هياكل البيانات الخطية باستخدام هيكل المصفوفة (array) . ولنأخذ مثالا عن طريقة البحث عن عنصر في مصفوفة او ما يسمى البحث الخطي (linear Search) :-

```
class ItemNotFoundException extends Exception {
item a [ ] = new a [ n ]
static int linearSearch ( item a [ ] , item x )
    throws ItemNotFoundException {
    for (int i= 0; i < a.length; i++)
        if ( x . equals ( a [ i ] ) return i ;
        throw new ItemNotFoundException ( ) ;
    }
    try {
        int found = linearSearch ( a , x )
        // do what ever necessary with a [ found ]
        ...
    }
    catch ( ItemNotFoundException e ) {
        // react to x not found
        ...
    }
}
```

وفي المثال اعلاه فنحن نرجع موقع العنصر داخل المصفوفة عند ايجاد العنصر وبعبسة يمكن أن نأخذ اي عمل استثنائي مثل إرجاع القيمة 1 - مثلاً ، ولنأخذ مثالا آخر نستخدم فيه المكنس لعكس اي سلسلة من الكلمات وفي هذه المرة سنستخدم المصفوفة Stack ، علما بأن هذه المصفوفة تحتوي على الاجراءات التالية (methods) :-

```

public class Stack extends Vector {
    public Object push ( Object item ) ;
    public Object pop ( ) throws EmptyStackException ;
    public Object peek ( ) throws EmptyStackException ;
    public boolean empty ( ) ;
    ...
}

import java . io . * ;
import java . util . * ;
class Stacker {
    public static void main ( String arg [ ] )
    throws IOException {
        DataInputStream in = new DataInputStream ( System in ) ;
        stack s = new Stack ( ) ;
        system . out . println ( " *** Testing the class stack ** " ) ;
        system . out . println ( " Type ^D or ^Z to end input " ) ;
        system . out . println ( " the original sentence is : " ) ;
        while ( true ) {
            try {
                String word = Text . readString ( in ) ;
                S.push ( word ) ;
            }
            catch ( EOFException e ) { break ; }
        }
        System . out . println ( " the reversed sentence is : " ) ;
        while ( ! s . empty ( ) )
            System . out . print ( s.pop ( ) + " " ) ;
        System . out . println ( ) ;
    }
}

```

أما هيكل البيانات الذي يسمى الطابور (queue) فصفاته كذلك هي أنه هيكل بيانات خطي بالإضافة إلى الصفات التالية :-

1. هناك نهائيات له مقدمة ومؤخرة (front and back) .
 2. الإضافة تكون على المقدمة .
 3. الحذف من المؤخرة .
 4. عملية استرجاع والوصول الى البيانات تتم من المقدمة والى المؤخرة .
- ويمكن أن يكون الطابور دائرياً (circular queue) .
 والمثال التالي يوضح كيفية تطبيق فكرة الطابور الدائري باستخدام هيكل المصفوفة (Array) :-

```

class Queue {
    Queue ( int m ) {
        if ( m <= maxQueue ) size = m ; else
            size = maxqueue ;
        front = 0 ;
        back = -1 ;
        reset ( ) ;
    }
    void add ( item ) throws QueueException {
        if ( live < size ) {
            back = ( back + 1 ) % maxQueue ;
            Q [ back ] = x ;
            live ++ ;
        }
        else throw new QueueException ( " full " ) ;
    }
    Object remove ( ) throws QueueException {
        if ( live > 1 ) {
            Object x = Q [ front ] ;
            front = ( front + 1 ) % maxQueue ;
            live -- ;
            return x ;
        }
        else throw new QueueException ( " Empty " ) ;
    }
    boolean empty ( ) { return live == 0 ;
    boolean full ( ) { return live == size ;

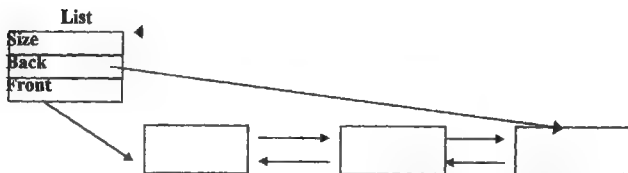
```

```

void rest () { now = front ; }
void succ () { now = ( now + 1 ) % maxQueue ; }
boolean eol () { if ( back == maxQueue )
    return now == 0 ;
    else return now > back ; }
Object current () { return Q [ now ] ; }
private
int size ,
    front , back ,
    live ,
    now ;
private Object Q [ ] = new Object [ maxQueue ] ;
static private Object int maxQueue = 100 ;
}
class QueueException extends Exception {
    QueueException ( String s ) { super ( s ) ; }
}

```

ويمكن تطبيق هياكل البيانات الخطية باستخدام القوائم (lists) وليس باستخدام المصفوفات (Arrays) ويعتبر هذا التطبيق ديناميكياً . والقوائم متعددة الانواع فهناك قوائم مربوطة بطريقة احادية (Single - linked) او ثنائية الربط (Doubly - linked) . والقوائم هي عبارة عن عقد (nodes) مربوطة ببعضها ليس عن طريق مؤشرات pointers بل عن طريق مرجع في الذاكرة (reference) والقوائم ثنائية الربط هي أكثر شيوعاً واستخداماً والشكل (1-6) يوضح طريقة الربط عن طريق مرجعية الذاكرة للقوائم ثنائية الربط .



شكل 6 - 1 :- شكل القائمة ثنائية الربط

وتعرف العقدة في القائمة ثنائية الربط كالآتي :-

```
class Node {  
    Node ( Object d , Node f , Node b ) {  
        data = d ;  
        forward = f ;  
        backward = ;  
    }  
    Node Forward , backward ;  
    Object data ;  
}
```

والبرنامج التالي يقوم ببناء اجراءات تعمل مع القائمة ثنائية الربط مثل اضافة عقدة أو حذف عقدة والبحث عن معلومة في عقدة معينة .

```
class List {  
    List ( ) { now = null ; front = null ; back = null ; size = 0 ; }  
    void addBefore ( Object x , Node pos ) {  
        if ( size == 0 ) {  
            front = new Node ( x , null , null ) ;  
            back = front ;  
        }  
        else if ( pos == null ) {  
            Node T = new Node ( x , null , back ) ;  
            back . forward = T ; ;  
            back = T ;  
        }  
        else if ( pos == front ) {  
            Node T = new Node ( x , front , null ) ;  
            front . backward = T ;  
            front = T ;  
        }  
        else {  
            Node T = new Node ( x . pos , pos . backward ) :  
            pos . backward . forward = T ;  
            pos . backward = T  
        }  
    }  
}
```

```

        size ++ ;
    }

    void addAfter ( Object x , Node pos ) {
        if ( size == 0 ) {
            front = new Node ( x , null , null ) ;
            back = front ;
        }
        else if ( pos == null ) || ( pos == back ) ) {
            Node T = new Node ( x , null , back ) ;
            back . forward = T ;
            back = T ;
        }
        else if ( pos = front ) {
            Node T = new Node ( x , pos . forward , pos ) ;
            pos . forward . backward = T ;
            pos . forward = T ;
        }
    }

    size ++ ;
}

Object remove ( Node pos )
    Object T = pos . data ;
    if ( front == back ) {
        front = null ;
        back = null ;
    } else
    if ( pos == front ) {
        front . forward . backward = null ;
        front = front . forward ;
    } else
    if ( pos == front ) {
        back . backward . forward = null ;
        back = back . backward ;
    } else
    pos . backward . forward = pos . forward ;

```

```

        pos . forward . backward = pos . backward ;
        size -- ;
        return T ;
    }

```

**Node search (Comparable x , int comp) throws
ItemNotFoundException {**

```

    if ( front == null ) return front ;
    for ( reset ( ) ; ! eol ( ) ; succ ( ) )
        switch ( comp )
            case Comparable . LESS :
                if ( x . less ( ( Comparable ) current ( ) ) ) return now ;
                break ;
            case Comparable . SAME :
                if ( x . same ( ( Comparable ) current ( ) ) ) return now ;
                break ;
            case Comparable . MORE :
                if ( ! x . less ( ( Comparable ) current ( ) ) ) return now ;
                break ;
        }
    if ( comp == Comparable . LESS ) return null ; else
    if ( comp == Comparable . MORE ) return null ;
    else throw ItemNotFoundException ( ) ;
}
boolean isempty ( ) { return front == null ; }
Object current ( ) { return now . data ; }
void rest ( ) { now = front ; }
boolean eol ( ) { return now . forward ; }
void succ ( ) { now . now . forward ; }
Node front , back ;
int size ;
private Node now
}
class ItemNotFoundException extends Exception { }

```


يمكننا أن نبني إجراءات كثيرة على الهياكل البيانات الخطية مثل المصفوفة والقائمة ، ولنأخذ على سبيل المثال بناء عملية ترتيب مصفوفة تحتوي على خيوط رمزية وبطريقة الترتيب بالانخال (insertion sort) .

```
public class InsertionSort {
    public static void main ( string [ ] args ) {
        String [ ] names = { " Cbs " , " Abc " } ;
        InsertionSort . sort ( names ) ;
        for ( int i= 0 ; i = names . length ; i++ ) {
            System . out . print ( names [ i ] + " " ) ;
        }
    }

    public Static void sort ( String [ ] string - array ) {
        int n = string - array . length ;
        for ( int i= n - 1 ; i > 0 ; i-- ) {
            String current = string - array [ i - 1 ] ;
            int j ;
            for ( j = i - 1 ; j = n - 1 & & current . compareTo ( string - array
                [ j + 1 ] ) > 0 ; j++ )
                string - array [ j ] = string - array [ j + 1 ] ;
            string - array [ j ] = current ;
        }
    }
}
```

6 - 2- هياكل البيانات اللاخطية :-

هي تلك الهياكل التي لا تكون عقدها متسلسلة بل لها تركيبية معينة مثل هيكل الشجرة (tree) والهياكل الشبكي (graph) ، وهذا النوع من الهياكل يطبق بصورة طبيعية من خلال استخدامنا الهيكل الديناميكي وليس باستخدام المصفوفات ، ولنأخذ مثالا على تمثيل هيكل الشجرة الثنائية المتفرع (Binary Tree) وبعض الإجراءات اللازمة لها مثل اضافة عقدة في الجانب الايسر او اليمين وتدقيق وجود عقدة في الاتجاه اليمين او الايسر وحذف عقدة من الجانب الايسر او اليمين في الشجرة .

```

import java . util . * ;
public class Binary Tree {
    public static void main ( String [ ] args ) {
        String [ ] names = { " Bbc " , " Abc " } ;
        Binary Tree stringTree = new Binary Tree ( names ) ;
        Enumeration nodes = stringsTree . elements ( ) ;
        while ( nodes . hasMoreElements ( ) ) {
            System.out . println ( Node ) nodes . nextElement ( ) . data ) ;
        }
    }
    public BinaryTree ( ) { }
    public BinaryTree ( Object [ ] elements ) {
        if ( elements . length == 0 ) return ;
        Node currentNode , tempNode ;
        currentNode = root = new Node ( Elements [ 0 ] ) ;
        Vector nodeQueue = new Vector ( ) ;
        for ( int i= 1 ; i< elements . length ; i + + ) {
            if ( currentNode . degree ( ) == 2 ) {
                currentNode = ( Node ) nodeQueue . firstElement ( ) ;
                nodeQueue . removeElementAt ( 0 ) ;
            }
        }
        if ( ! currentNode . hashLeftChild ( ) )
            tempNode = currentNode . addLeftChild ( elements [ i ] ) ;
        else tempNode = currentNode . addRightchild ( elements [ i ] ) ;
        nodeQueue . addElement ( tempNode ) ;
    }
}

class Enumeration elements ( ) {
    return ( this . new Traversal ( ) ) ;
}

class Traversal implements Enumeration {
    private Vector nodes ;
    public Traversal ( ) {
        nodes = new Vector ( ) ;
    }
}

```

```

        if ( root != null )
            nodes . addElement ( root ) ;
    }
    public boolean hasMoreElements ( ) {
        if ( nodes . size ( ) == 0 )
            return false ;
        return true ;
    }
    public Object nextElements ( ) {
        Node tempNode = ( Node ) nodes . elementAt ( 0 ) ;
        nodes . removeElementAt ( 0 ) ;
        if ( tempNode . hasLeftChild ( ) )
            nodes . addElement ( tempNode . getLeftChild ( ) ) ;
        if ( tempNode . hasRightChild ( ) )
            nodes . addElement ( tempNode . getRightChild ( ) ) ;
        return tempNode ;
    }
}
class Node {
    public Node ( Object data ) { this . data = data ; }
    public void setData ( Object data ) {
        this data = data ; }
    public Object getdata ( ) { return data ; }
    public boolean hasLeftChild ( ) {
        return ( rightChild != null ) ;
    }
    public boolean hasLeftChild ( ) {
        return ( rightChild != null ) ;
    }
    public Node getLeftChild ( ) {
        return leftChild ; }
    public Node getRightChild ( ) {
        return RightChild ; }
    public Node addLeftChild ( Object data ) {
        Node tempNode = new Node ( data ) ;
        leftChild = tempNode ;
        return tempNode ; }
}

```

```

public Node addRightChild ( Object data ) {
    Node tempNode = new Node (data ) ;
    rightChild = tempNode ;
}
public Node deleteLeftChild ( ) {
    Node tempNode =leftChild ;
    leftChild = null ;
    return tempNode ; }
public int degree ( ) {
    int i = 0 ;
    if ( leftChild != null ) i ++ ;
    if (RightChild != null ) i ++ ;
    return i }
private Object data ;
private Node leftChild , rightChild ;
}
Node root ;
}

```

والآن نقوم ببناء بعض العمليات على هيكل الشجرة الذي قمنا ببنائه . والمثال التالي ينفذ زيارة عقد الشجرة بطريقة (Preorder Traversal) .

```

class PreorderTraversal implements Enumeration {
    private Stack nodes ;
    public PreorderTraversal ( ) {
        nodes = new Stack ( ) ;
        if ( root != null ) nodes . push ( root ) ;
    }

    public boolean hasMoreElements ( ) {
        if ( nodes . size ( ) == 0 ) return false ;
        return true ;
    }

    public Object nextElement ( ) {
        Node currentNode = ( Node ) nodes . peek ( ) ;
        nodes . pop ( ) ;
    }
}

```

```

        if ( currentNode . hasRightChild ( ) )
            nodes . push ( currentNode . RightChild ) ;
        if ( currentNode . hasLeftChild ( ) )
            nodes . push ( currentNode . leftChild ) ;
        return currentNode ;
    }
}

```

ولنأخذ مثال آخر على بناء إجراءات أخرى على هيكل الشجرة ولنأخذ مثال عن البحث في الشجرة الثنائية :-

```

import java . util . * ;
public class BinarySearchTree extends BinaryTree {
    public static void main ( String [ ] args ) {
        String [ ] names = { " Cbc " , " Bbc " , " Abc " } ;
        BinarySearchTree stringTree = new BinarySearchTree ( ) ;
        for ( int i = 0 ; i < names . length ; i ++ )
            stringTree . addData ( names [ i ] ;
        Enumeration nodes = stringTree . elements ( ) ;
        while ( nodes . hasMoreElements ( ) ) {
            System . out . println ( ( Node ) nodes . NextElement ( ) . getData ( ) ) ;
        }
    }
    public BinarySearchTree ( ) { }
    public void addData ( String s ) {
        if ( root == null ) root = new Node ( s ) ;
        else addData ( root , s ) ;
    }
    private void addData ( Node currentNode , String s ) {
int c = ( ( String ) currentNode . getData ( ) ) . compareTo ( s ) ;
        if ( c == 0 ) return ;
        else if ( c > 0 ) {
            if ( currentNode . hasLeftChild ( ) )
                addData ( curenentNode . getLeftChild ( ) , s ) ;
            else currentNode . addLeftChild ( s ) ;
        }
    }
}

```

```
else {  
    if ( currentNode . hasRightChild ( ) )  
        addData ( currentNode . getRightChild ( ) , s ) ;  
    else currentNode . addRightChild ( s ) ;  
}
```

الفصل السابع
التعامل مع الحركة
في صفحات الويب

الفصل السابع

التعامل مع الحركة في صفحات الويب

7-1 التعامل مع الصور المتحركة :-

أما أكثر أنواع الحركة التي يمكن إظهارها على صفحات الويب هي باستخدام حركة الصور ، ولكي نستعرض الأساليب والتقنيات اللازمة لحركة الصور ، أولاً نذكر كيفية عرض صورة على الشاشة، والمثال التالي يوضح عرض صورة باسم "ladybug.gif" على صفحة ويب وبدون حركة:

```
1: import java.awt.Graphics;
2: import java.awt.Image;
4: public class Ladybug extends java.applet.Applet {
6:     Image bugimg;
8:     public void init () {
9:         bugimg = getImage(getCodeBase( ,
10:             " images / ladybug . gif " ) );
11:     }
13:     public void paint ( Graphics g ) {
14:         g . drawImage ( bugimg , 10 , 10 , this ) ;
15:     }
16: }
```

وبلاحظ أن عملية رسم الصورة في صفحة ويب يتطلب استدعاء الإيعاز (drawImage) الذي فيه ابعاد الزاوية اليسرى للصورة بعدد الصفوف والأعمدة ، ولكن عند عرض الصورة يمكن التحكم بمعجم الصورة التي يتم عرضها من خلال استخدام إيعاز (getWidth) والبرنامج التالي يقوم بعرض نفس الصورة السابقة بأربعة أحجام مختلفة .

```

1: import java . awt . Graphics ;
2: import java . awt . Image ;
4: public class ladyBug2 extends java . applet . Applet {
6:     Image bugimg ;
8:     public void init ( ) {
9:         bugimg = getImage ( getCodeBase ( ) ,
10:            " images / ladybug . gif " ) ;
11:     }
13:     public void paint ( Graphics g ) {
14:         int iwidth = bugimg . getWidth ( this ) ;
15:         int iheight = bugimg . getHeight ( this ) ;
16:         int xpos = 10 ;
18:         // 25 %
19:         g . drawImage ( bugimg , xpos , 10
20:            iwidth / 4 , iheight / 4 this ) ;
22:         // 50 %
23:         xpos + = ( iwidth / 4 ) + 10 ;
24:         g . drawImage ( bugimg , xpos , 10 ,
25:            iwidth / 2 . iheight / 2 , this ) ;
26:
27:         // 100 %
28:         xpos + = ( iwidth / 2 + 10 ;
29:         g . drawImage ( bugimg , xpos , 10 , this ) ;
30:
31:         // 150 % x , 25 % y
32:         g . drawImage ( bugimg , 10 , iheight + 30 ,
33:            ( int ) ( iwidth / 4 , this ) ;
34:     }
35: }

```

والآن لنأخذ مثالاً على تحريك صورة معينة (tt2gif) وبصورة احتياطية داخل الشاشة ،
والمثال التالي يعبر عن تصميم لعبة بين المستخدم الذي يقوم بملاحقة الصورة المتحركة ، فإذا

ضرب مؤشر الفأرة عليها يسجل له نقطة وبكسة يحصل على عبارة فشل في إصابة الصورة المتحركة :-

```
import java . awt . * ;
import java . applet . * ;
public class putis extends Applet {
int x , y , limitx , limity ;
int wins ;
int d ;
int change ;
Image duke ;

public void init ( ) {
wins = 0 ;
d = size ( ) . width - 1 ;
change = d - d / 10 ;
duke = getImage ( getCodeBase ( ) , ' T2 . gif ' ) ;
}

public void paint ( Graphics g ) {
limitx = duke . getWidth ( this ) ;
limity = duke . getHeight ( this ) ;
g . drawRect ( 0 , 0 , d , d ) ;
x = ( int ) ( Math . random ( ) * 1000 ) % change ;
y = ( int ) ( Math . random ( ) * 1000 ) % change ;
g . drawImage ( duke , x , y , this ) ;
}

public boolean mouseMove ( Event evt , int mx , int my )
{
if (( mx % 3 == 0 ) && ( my % 3 == 0 ))
repaint ( ) ;
return true ;
}
```

```

public boolean mouseDown ( Event evt , int mx , int my ) {
    requestFocus ( ) ;
    if (( x < mx && mx < x + limitx ) && ( y < my && my < y + limity ))
    {
        wins ++ ;
        getAppletContext ( ) . showStatus ( ' Caught it ! Total ' + wins ) ;
        play ( getCodeBase ( ) , ' cowbell . au ' ) ;
    }
    else
    {
        getAppletContext ( ) . showStatus ( ' Missed Again . ' )
        play ( getCodeBase ( ) , ' welcome . au ' ) ;
    }
    repaint ( ) ;
    return false ;
}

```

والآن وبعد أن تعلمنا كيفية تحريك صورة معينة ، يمكننا التحول الى دراسة أساليب تحريك الصور المتعددة ، وبسط طريقة هي من خلال عرض أكثر من صورة بتوقيت زمني معين ، وهذه الصور يجب أن يكون علاقة بينهما كان تكون نفس الصورة مع اجراء بعض التغيرات فيها ، والمثال الذي نريد استخدامه لتوضيح هذه التقنية التي نطلق عليها الحركة بتقليب الصورة (Animation using Image flipping) ، وأول ما نحتاجه هو قيمة عدة صور فيها تغيرات معينة وعند عرضها الواحدة بعد الاخرى او بأي ترتيب معين يمكن أن نظهر من خلال هذا العرض حركة معينة ، ولناخذ على سبيل المثال تسعة صور لقط يؤدي حركات معينة مثل الركض والجلوس والنوم وكما يوضحه الشكل (1-7) :



شكل 7-1 : صور مختلفة للقط الملقب بـ (Neko) .

وطريقة وضع الصور في مكس أو مصفوفة تكون من خلال التعريف التالي :-

Image nekopies [] = new Image [9] ;

Image currentimg ;

ونحتاج الى بناء اجراء لتهيئة تحميل الصور باسم (init) ، وبعد ذلك فان عملية

تقليب الصور تتم من خلال اجراء (run) الذي يقوم بعرض صورة القطعة تركض من

اليسار الى اليمين وتتوقف في الوسط وتقرش جلدها كذلك اربع مرات ثم تمام ، واخيراً تنتهي

وتركض الى يمين الشاشة ، وعملية جعل القطعة تمام هو من خلال استدعاء اجراء التأخير (

pause) وتفاصيل هذا البرنامج ندرجها كالتالي :-

36: import java . awt . Graphics ;

37: import java . awt . Image ;

38: import java . awt . Color ;

40: public class Neko extends java.applet.Applet

41: implements Runnable {

43: Image nekopies [] = new Image [9] ;

44: Image currentimg ;

45: Thread runner ;

46: int xpos ;

47: int ypos = 50 ;

49: public void init () {

50: String nekosrc [] = { 'right1 . gif' , ' right2 . gif' ,

51: ' stop . gif' , "yawn . gif" , 'scratch1 . gif' ,

52: ' scratch2 . gif' , 'sleep1 . gif' , "sleep2 . gif" ,

53: ' awake . gif' } ;

```

55:         for ( int i = 0 ; i < nekopics . length ; i ++ ) {
56:             nekopics [ i ] = getImage ( getCodeBase ( ) ,
57:                 ' images / ' + nekosrc [ i ] ) ;
58:         }
59:     public void start ( ) {
60:         if ( runner == null ) {
61:             runner = new Thread ( this ) ;
62:             runner . start ( ) ;
63:         }
64:     }
65: }
66: public void stop ( ) {
67:     if ( runner != null ) {
68:         runner . stop ( ) ;
69:         runner = null ;
70:     }
71: }
72: }
73:
74: public void run ( ) {
75:     setBackground ( Color . white ) ;
76:     // run from one side of the screen to the middle
77:     nekorun ( 0 , this . size ( ) . width / 2 ) ;
78:     // stop and pause
79:     currentimg = nekopics [ 2 ] ;
80:     repaint ( ) ;
81:     pause ( 1000 ) ;
82:     // yawn
83:     currentimg = nekopics [ 3 ] ;
84:     repaint ( ) ;
85:     pause ( 1000 ) ;
86:     // scratch four times
87:     nekoscratch ( 4 ) ;
88:     // sleep for 5 ' turns '
89:     nekosleep ( 5 ) ;
90:     // wake up and run off
91:     currentimg = nekopics [ 8 ] ;

```

```

99:         repaint ( ) ;
100:         pause ( 500 ) ;
101:         nekorun ( xops , this . size ( ) . width + 10 ) ;
102:     }
103:
104: void nekorun ( int start , int end ) {
105:     for ( int i = start ; i < end ; i += 10 ) {
106:         this . xpos = i ;
107:         // swap images
108:         if ( currentimg == nekopics [ 0 ] )
109:             currentimg = nekopics [ 1 ] ;
110:         else if ( currentimg == nekopics [ 1 ] )
111:             currentimg = nekopics [ 0 ] ;
112:         else currentimg = nekopics [ 0 ] ;
113:
114:         repaint ( ) ;
115:         pause ( 150 ) ;
116:     }
117: }
118:
119: void nekoscratch ( int numtimes ) {
120:     for ( int i = numtimes ; i > 0 ; i -- ) {
121:         currentimg = nekopics [ 4 ] ;
122:         repaint ( ) ;
123:         pause ( 150 ) ;
124:         currentimg = nekopics [ 5 ] ;
125:         repaint ( ) ;
126:         pause ( 150 ) ;
127:     }
128: }
129:
130: void nekosleep ( int numtimes ) {
131:     for ( int i = numtimes ; i > 0 ; i -- ) {
132:         currentimg = nekopics [ 6 ] ;
133:         repaint ( ) ;
134:         pause ( 250 ) ;

```

```

135:         currentimg = nekopics [ 7 ] ;
136:         repaint ( ) ;
137:         pause ( 250 ) ;
138:     }
139:
140:     void pause ( int time ) {
141:         try { Thread . sleep ( time ) ; }
142:         catch ( InterruptedException e ) { }
143:     }
144:
145:     public void paint ( Graphics g ) {
146:         g . drawImage ( currentimg , xpos , ypos , this ) ;
147:     }
148: }

```

وبالرغم من أن الحركة ستظهر من خلال تقلب الصور إلا أنها ستظهر مرتجفة نوعاً ما (flicker) ولأجل تقليل عملية ارتجاج عرض الصورة هناك عدة تقنيات يمكن أن تقلل من عملية الارتجاج هذه أهمها تقنية تسمى التخزين المضاعف (Double Buffering) ، حيث نقوم باستدعاء اجراء رسم الصورة (paint) خارج الشاشة (offscreen) على سطح او ذاكرة (surface) معين ومن ثم عرضها مرة واحدة وبصورة كاملة على الشاشة وهذا سيقبل بكل تأكيد عملية ارتجاج الصورة ، وعملية رسم الصورة خارج الشاشة يعني وجود عملية اخرى متزامنة مع ما يجري على الشاشة وهذا يعني وجود خيوط تنفيذية متعددة داخل البرنامج (multithread) ، والبرنامج التالي يوضح تطبيقاً لتقنية التخزين المضاعف :-

```

import java . awt . * ;
import java . applet . * ;
import java . awt . image . * ;
public class Animate extends Applet implements Runnable {
    int cardWidth = 43 , cardHeight = 61 , imgCt = 50 , thisPos ;
    Image currImg , thisCard [ ] = new Image [ imgCt ] , winScratch ;
    Graphics gScratch ;
    Thread runner ;
    public void init ( ) {

```



```

int randomCard , cardPos ; i ;
Image playingCards ;
ImageFilter cardFilter ;
ImageProducer cardProducer ;
boolean usedCards [ ] = new boolean [ 52 ] ;
winScratch = createImage ( this . size ( ) . width , this . size . height ) ;
gScratch = winScratch . getGraphics ( ) ;
setBackground ( Color . white ) ;
playingCards = getImage ( getCodeBase ( ) . ' cards . gif ' ) ;
for ( i = 0 ; i < imgCt ; i ++ ) {
    randomCard = ( int ) java . lang . Math . random ( ) * 52 ) ;
    if ( usedCards [ randomCard ] ) {
        i - ;
    }
    else {
        cardPos = ( reandomCard * cardWidth ) + i ;
        cardFilter = new CropImageFilter ( cardPos , i ,
            cardWidth , cardHeight ) ;
        cardProducer = new FilteredImageSource ( playingCards .
            perSource ( ) , cardFilter ) ;
        thisCard [ i ] = createImage ( cardProducer ) ;
        usedCards [ reandomCard ] = true ;
    }
}

thisPos = ( int ) ( java . lang . Math . random ( ) * 200 ) ;
currImg = thisCard [ i ] ;
}
public void start ( ) {
    .runner = new Thread ( this ) ;
    runner . start ( ) ;
}

```

```

public void run () {
    for ( int i = 1 ; imgCt ; i++ ) {
        try {
            Thread . sleep ( 200 ) ;
        }
        catch ( InterruptedException e ) {
        }
        thisPos = ( int ) ( java . lang . Math . random ( ) * 200 ) ;
        currImg = thisCard [ i ] ;
        repaint ( ) ;
    }

    public void paint ( Graphics g ) {
        gScratch . setColor ( this . getBackground ( ) ) ;
        gScratch . fillRect ( 0 , 0 , this . size ( ) . width , this . size ( ) . height ) ;
        gScratch . setColor ( Color . black ) ;
        gScratch . drawImage ( currImg , thisPos + 3 , 3 , this ) ;
        gScratch . drawRoundRect ( thisPos + 1 , 1 , cardWidth + 2 , 5 , 5 ) ;
        g . drawImage ( winScratch , 0 , 0 , this ) ;
    }

    public final void update ( Graphics g ) {
        paint ( g ) ;
    }
}

```

7-2 الحركة باستخدام التقييط :-

الحركة يمكن توليدها من خلال توليد نقاط وبألوان متعددة في مناطق مختلفة مسن الشاشة ،
والمثال التالي يقوم باستخدام حركة جهاز الفأرة وعند كل كبسه يتم رسم نقطة على الشاشة .

```

1: import java . awt . Graphics ;
2: import java . awt . Color ;
3: import java . awt . Event ;

```

```

5: public class Spots extends java.applet.Applet {
7: final int MAXSPOTS = 10 ;
8:     int xspots [] = new int [ MAXSPOTS ] ;
9:     int yspots [] = new int [ MAXSPOTS ] ;
10:    int currspots = 0 ;
12:    public void init () {
13:        setBackground ( Color . white ) ;
14:    }
15:
16:    public boolean mouseDown ( Event evt , int x , int y ) {
17:        if ( currspots < MAXSPOTS )
18:            addspot ( x , y ) ;
19:        else System . out . println ( ' Too many spots . ' ) ;
20:        return true ;
21:    }
22:
23:    void addspot ( int x , int y ) {
24:        xspots [ currspots ] = x ;
25:        yspots [ currspots ] = y ;
26:        currspots ++ ;
27:        repaint () ;
28:    }
29:
30:    public void paint ( Graphics g ) {
31:        g . setColor ( Color . blue ) ;
32:        for ( int i = 0 ; i < currspots ; i ++ ) {
33:            g . fillOval ( xspots [ i ] -10 , yspots [ i ] -10 , 20 , 20 ) ;
34:        }
35:    }
36: }

```

ولكن هذا البرنامج بسيط لا يوفر حركة حقيقية ولكن يمكن تطوير هذا البرنامج لتوليد تنقيط بطريقة تشبه توليد أعضاء الاحتفالات الرسمية في سماء الليل مثلاً .

7-3 الحركة باستخدام الخطوط :-

الحركة باستخدام الخطوط (lines) أكثر شيوعاً من الحركة بواسطة التقيط والمثال التالي يمثل عملية رسمنا لخط بواسطة جهاز الفأرة ومن ثم يقوم البرنامج بتحريك هذا الخط مولداً حركة ذات تأثير مرئي معين .

```
1: import java . awt . Graphics ;
2: import java.awt.Color;
3: import java . awt . Event ;
4: import java . awt . Point;
6: public class Lines extends java . applet . Applet {
8: final int MAXLINES = ;
9: Point starts [ ] = new Point[MAXLINES] ; // starting points
10: Point ends [ ] = new point [ 10 ] ; // endingpoints
11: Point anchor ; // start of current line
12: Point currentpoint ; // current end of line
13: int currline = 0 ; // number of lines
15: public void init ( ) {
16: setBackground ( Color . white ) ;
17: }
18:
19: public boolean mouseDown ( Event evt , int x , int y ) {
20: anchor = new Point ( x , y ) ;
21: return true ;
22: }
23:
24: public boolean mouseUp ( Event evt , int x , int y ) {
25: if ( currline < MAXSPOTS )
26: addline ( x , y ) ;
27: else System . out . println ( " Too many lines . " ) ;
28: return true ;
29: }
30:
31: public boolean mouseDrag ( Event evt , int x , int y ) {
32: currentpoint = new Point ( x , y ) ;
```

```

33:         repaint ( ) ;
34:         return true ;
35:     }
36:
37:     void addline ( int x , int y ) {
38:         starts [ currline ] = anchor ;
39:         ends [ currline ] = new Point ( x , y ) ;
40:         currline ++ ;
41:         currentpoint = null ;
42:         repaint ( ) ;
43:     }
44:
45:     public void paint ( Graphics g ) {
46:
47:         // Draw existing lines
48:         for ( int i = 0 ; i < currline ; i ++ ) {
49:             g . drawLine ( starts [ i ] . x , ends [ i ] . y ,
50:                           ends [ i ] . x , ends [ i ] . y ) ;
51:         }
52:
53:         // draw current line
54:         g . setColor ( Color . blue ) ;
55:         if ( currentpoint != null )
56:             g . drawLine ( anchor . x , anchor . y ,
57:                           currentpoint . x , currentpoint . y ) ;
58:         }
59:     }

```

4-7 الحركة باستخدام الحروف -

يمكن توليد حركة معينة على الشاشة باستخدام الحروف ، والمثال التالي ينم فيه تحريك أي حرف تقوم بطبعه من خلال لوحة المفاتيح .

```

1: import java . awt . Graphics ;
2: import java . awt . Event ;
3: import java . awt . Font ;
5: public class Keys extends java . applet . Applet {

```

```

7:      char currkey ;
8:      int currx ;
9:      int curry :
11:     public void int ( ) {
12:         currx = ( this . size ( ) . width / 2 ) - 8 ; // default
13:         curry = ( this . size ( ) . height / 2 ) - 16 ;
15:         setBackground ( Color . white ) ;
16:         setFont( new Font ( ' Helvetica ' , Font . Bold , 36 ) ) ;
17:     }
19:     public boolean keyDown ( Event evt , int key ) {
20:         switch ( key ) {
21:             case Event . DOWN ;
22:                 curry + = 5 ;
23:                 break ;
24:             case Event . UP ;
25:                 curry - = 5 ;
26:                 break ;
27:             case Event . LEFT ;
28:                 currx - = 5 ;
29:                 break ;
30:             case Event . RIGHT :
31:                 currx + = 5 ;
32:                 break ;
33:             default ;
34:                 currkey = ( char ) key ;
35:             }
37:             repaint ( ) ;
38:             return true ;
39:         }
41:         public void paint ( Graphics g ) {
42:             if ( currkey != 0 ) {
43:                 g.drawString ( String . valueOf ( currkey ) , currx , curry ) ;
44:             }
45:         }
46:     }

```

5-7 الحركة باستخدام الارقام :-

واخيراً يمكن إظهار حركة باستخدام تغير في الارقام على الشاشة كما في مثال إظهار الساعة الرقمية على الشاشة :

```
1: import java . awt . Graphics ;
2: import java . awt . Font ;
3: import java . awt . Date ;
5: public class DigitalClock extends java . applet . Applet {
7: Font theFont = new Font ( "TimesRoman",Font.BOLD,24 ) ;
8:     Date theDate ;
10:     public void start ( ) {
11:         while ( true ) {
12:             theDate = new Date ( ) ;
13:             repaint ( ) ;
14:             try { Thread . sleep ( 1000 ) ; }
15:             catch ( InterruptedException e ) { }
16:         }
17:     }
19:     public void paint ( Graphics g ) {
20:         g . setFont ( theFont ) ;
21:         g . drawString ( theDate . toString ( ) , 10 , 50 ) ;
22:     }
23: }
```

المصادر و المراجع

المصادر والمراجع

1. Java Gently: Programming Principles Explained
J. M. Bishop , Addison - Wesley , 1997 .
2. The complete guide to Java , A. Corbly , Computer Step , 1997 .
3. The Java Programming Language , K. Arnold and J. Gosling ,
Addison - Wesley , 1996 .
4. The Java Tutorial :Object-Oriented Programming for the Internet ,
M. Campione and K. Walrth , Addison - Wesley , 1997 .
5. The Java Class Libraries : An Annotated Reference
P. Chan and R. Lee , Addison - Wesley , 1997 .
6. The Java Language Specification ,
J. Gosling , B. Joy and G. Steele , Addison -Wesley , 1997 .
7. Concurrent Programming in Java : Design Principles and Patterns ,
D. Lea , Addison - Wesley , 1997 .
8. Java Essentials for Card C ++ Programmers ,
B. Boone , Addison - Wesley , 1996 .
9. Java Data Structures and Programming , Liwa Li , Springer , 1998 .
10. Essential Java , J. Manger , McGraw - Hill , 1996 .
11. Advanced Java I. I Programming , J. Rice and I. SalisBury ,
McGraw - Hill , 1997 .

Internet Programming using JAVA

Dr. Sabah M.A. MOHAMAD
Ph D, MBCS, MIEF, VMACM
Associate Professor, Chirman

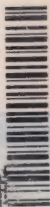
Dr. Jinan A.W.FAIDHI
Ph D, MBCS, VMACM
Associate Professor

Applied Science University

ISBN 9957 - 400 - 09 - 6 (ردمك)



Bibliotheca Alexandrina



0600932

مؤسسة الوراق للخدمات الحديثة

عمان - شارع الجامعة الأردنية - عمارة العساف

ص.ب ١٥٢٧ عمان ١١٩٥٣ - الأردن

تلفكاس ٥٣٣٧٧٩٨